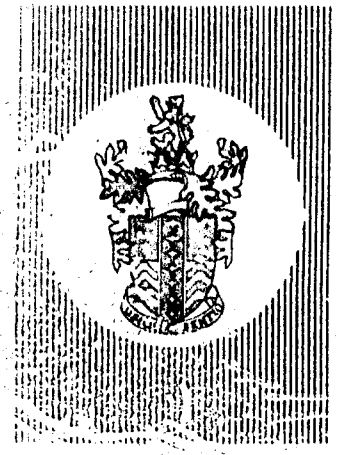


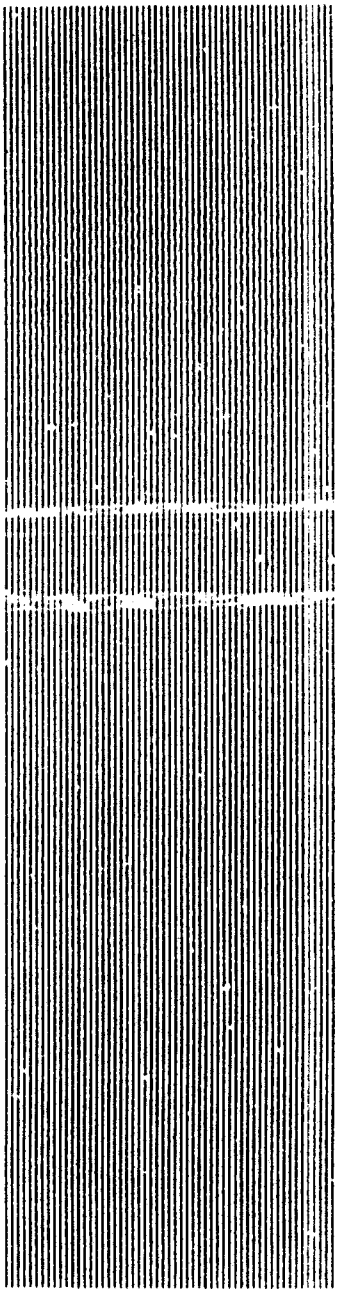
Report No. 88002

ROYAL SIGNALS AND RADAR ESTABLISHMENT
MALVERN

FORMAL SPECIFICATION OF A SECURE DOCUMENT CONTROL SYSTEM FOR SMTE

Author: C L Harrold

PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.



February 1988

UNLIMITED

ROYAL SIGNALS AND RADAR ESTABLISHMENT

REPORT 88002

Title: Formal Specification of a Secure Document Control System for SMITE

Auther: C L Harrold

Date: February 1988

SUMMARY

This paper formally describes the requirements for a demonstration of a secure electronic registry control system (Sercus) to be implemented using the security attributes of the SMITE secure capability computer.

Contents

1. Introduction

This document formally describes the requirements for a demonstration of a secure electronic registry control system (Sercus) to be implemented using the security attributes of the SMITE secure capability computer [1,2,3]. The formal notation used in this specification is 'Z', which has been developed by the Programming Research Group at Oxford University [4].

Sercus is intended to control the access to, and creation of, classified documents and files. In the paper world, all documents and files are centrally recorded and information as to who has had access to them is maintained. Sercus will enforce similar mechanisms. In addition to handling documents and files, users of Sercus will be able to send simple mail messages.

When users are logged on to Sercus they are presented with a display that consists of a set of windows. All the window software will be completely trustworthy, but may be used to invoke untrusted software. While untrusted software is active in a window the classification of the data is prominently displayed. Sercus will monitor the movement of objects between these windows, and correctly maintain the classification levels.

The specification of Sercus is divided into two distinct parts. The first part specifies the underlying components of the system, such as the documents, users and displays, and defines the basic operations on them. The second part shows how these components are combined together and constraints added to the basic operations to create a system that is both functional and secure.

This specification will form the basis of an investigation into techniques for proving conformance between high level specification and code level implementation. This is required to achieve the highest possible levels of design assurance.

Part 1

The following sections describe the various components of Sercus, such as documents, files and users, and define the simple operations upon them.

2. General Types

[CLASS]
There is a given set of classifications. These are used for classifying objects and giving clearances to users.

There is a relationship, ≥, between classifications to indicate domination. This will be needed when checking whether a user is cleared to see a document. For example, a user cleared to 'Secret' will be able to read a 'Restricted' document, because the classification Secret dominates Restricted. However, users cleared to Restricted will not be able to read Secret documents.

$$\_ \geq \_ : \mathit{partial\_order}(\ CLASS\ )$$

This ordering is partial because not all classifications are comparable. For example 'Secret UK Eyes Only' neither dominates nor is dominated by 'Secret US Eyes Only'.

It is useful to define a classification that is dominated by all other possible classifications.

$$bottom : CLASS$$

$$\forall\ class : CLASS \bullet class \geq bottom$$

There is a least upper bound operator between two classifications. This returns the lowest classification which dominates both of them. For example, a document that contains both Secret Atomic and Secret Nato information must be classified to at least 'Secret Atomic Nato'.

$$\_\ lub\ \_ : (\ CLASS \times CLASS\ ) \rightarrow CLASS$$

$$\forall\ a, b : CLASS \bullet\ a\ lub\ b \geq a \wedge a\ lub\ b \geq b \wedge \forall\ l : CLASS \mid l \geq a \wedge l \geq b \bullet l \geq a\ lub\ b$$

Note that this function is total. This implies that there is one classification which dominates all other possible classifications.

An operation that takes a set of classifications and returns the lowest classification that dominates all of them, is also defined.

$$LUB\_ : \mathbb{P}\ CLASS \rightarrow CLASS$$

$$\forall\ set : \mathbb{P}\ CLASS \bullet \forall\ c : set \bullet LUB\ set \geq c \wedge \forall\ l : CLASS \mid \forall\ c : set \bullet (\ l \geq c\ ) \Rightarrow (\ l \geq LUB\ set\ )$$

[STR]
There is a given set of character strings. These are intended to represent printed messages on the screen, such as the current classification. An operation which takes a classification and returns the string representing it is defined.

| show _ : CLASS ↣ STR

This is a function because there is only one string to represent each classification. The function is total because all classifications must be representable, and one to one because all the strings must be unique.

[CAPABILITY]
There is a given set of capabilities for all possible objects. A capability is a universal, unforgable name for an object. Capabilities are the means of refering to objects. Each capability refers to only one object, and there is only one capability for each object.

Capabilities are only distinguishable by the virtue of the objects they refer to. This means that whenever capabilities are copied it is impossible to tell them apart as they will refer to the same object. Hence, the restriction that there is a single capability for each object in Sercus, and consequently capabilities can be in more than one place at a time.

[TIME]
Time exists, and a relationship between two times to indicate 'not later than' is defined.

| _≤_ : total_order(TIME)

This ordering is total since all times are comparable.

The current time will always be available in Sercus, although no operations to update it will actually be defined.

| time_now : TIME

3. Documents and Files

3.1 Introduction

Sercus is intended to control the access to, and creation of, classified documents and files, and to model some of the mechanisms in use in the paper world that do this. In the paper world, all documents are centrally recorded on a Classified Document Register (cdr) and all files on a Filelist. Documents can be uniquely identified by their cdr number, or if they have been filed, by a file number and enclosure number. The cdr also records the people who have seen the document.

For the purposes of specification it is useful to partition the set of all possible capabilities into three sets.

```
caps_for_docs : ℙ CAPABILITY
caps_for_files : ℙ CAPABILITY
caps_for_anon : ℙ CAPABILITY
```

```
⟨ caps_for_docs, caps_for_files, caps_for_anon ⟩
                                        partition CAPABILITY
```

This means that when creating new documents, for example, it will not be necessary to check whether the capability you are giving the document already exists as a file or other capability.

3.2 Documents in Sercus

Documents are unalterable objects which may only be read. They have a classification bound into them and can be uniquely identified by a cdr number. The characters that make up the text of a document are uninteresting as far as specification at this level is concerned. The only interesting property of the contents of a document is that it may contain capabilities for other documents.

Documents do not move about the system, but are accessed via capabilities. Hence, unlike in the paper world the cdr will not record the location of a document, but will record which users have opened the document (note that simple possession of a capability does not automatically allow access and consequently it is not useful to record this fact). This mechanism will be added at a later date, see journalling in section 9.1 and 10.

[CDR_NUM]

There is a given set of cdr numbers that will be used to uniquely identify the documents. Cdr numbers are ordered, in the paper world on date of creation, and an operation to discover this order is defined.

```
_≥_ : total_order( CDR_NUM )
```

This ordering is total because all cdr numbers are comparable.

DOCUMENT

```
classification : CLASS
contents : ℙ CAPABILITY
cdr_number : CDR_NUM
```

```
contents ⊆ caps_for_docs   (1)
```

(1) Documents can only contain capabilities for other documents.

A mapping between document capabilities and their associated documents is required. Only some of the total set of capabilities for documents refer to existing documents, as other capabilities will be required whenever new documents are created.

THE_DOCS

```
which_doc : CAPABILITY ⇸ DOCUMENT                                    (1)
```

```
dom which_doc ⊆ caps_for_docs

∀ c : dom which_doc •
   which_doc( c ).contents ⊆ dom which_doc                           (2)

∀ i, j : dom which_doc •
   which_doc(i).cdr_number = which_doc(j).cdr_number ⇔ i = j         (3)
```

The which_doc function maps the document capabilities to their documents. The domain of this function is the set of all the document capabilities so far created. Hence the range is the set of all the documents created so far.

(1) This is a function because capabilities can only refer to one document. The function is one to one because there is only one capability for each document, and partial because not all the possible documents have yet been created.

(2) All the documents in Sercus may only contain capabilities for other documents that exist.

(3) Documents are uniquely identifiable by their cdr number.

```
ΔTHE_DOCS ≙ [ THE_DOCS'; THE_DOCS ]

ΞTHE_DOCS ≙ [ ΔTHE_DOCS | θTHE_DOCS' = θTHE_DOCS ]
```

3.3. Files in Sercus

Files in Sercus are not necessary for naming or accessing documents. They are simply a convienient grouping of document capabilities. In the paper world filed documents can be identified by a file and enclosure number. In order to be able to do this in Sercus, it is necessary to insist that the document capabilities may be on at most one file.

> Note that although documents can only be on a single file, they can contain capabilities for documents on other files.

[FTITLE]
All files will have a title, which is taken from the given set, and is itself classified. Files also have an overall classification, which must dominate the

title classification, and the classification of all the documents they contain. However, the overall classification is not necessarily the least upper bound of the title classifications and the document classifications, but could be significantly greater. For example, a Secret file may only contain Confidential documents. Documents on a file are ordered by enclosure numbers, which are taken from the set of natural numbers.

FILE

```
title : FTITLE
title_class, overall_class : CLASS
docs : seq CAPABILITY
```

```
ran docs ⊆ caps_for_docs                                    (1)

overall_class ≥ title_class                                 (2)

∀ i,j : dom docs • docs( i ) = docs( j ) ⇔ i = j            (3)
```

(1) Only capabilities for documents may be put on a file.

Note that although this specifies that files can contain only document capabilities, it is only later, see section 3.4, the Filing System, that it can be insisted that these documents actually exist.

(2) The overall classification of a file must dominate the title classification.

Note that the fact that the overall classification must also dominate the classification of all the documents on the file cannot be specified until later (section 3.4).

(3) Documents cannot be put on a file more than once.

Note that the position in the sequence of filed capabilities represents the enclosure number of the document.

As for documents, a mapping between the file capabilities and files is required.

THE_FILES

```
which_file : CAPABILITY ⇸ FILE
```

```
dom which_file ⊆ caps_for_files

∀ file : ran which_file • file.docs ≠ ⟨⟩                    (1)

∀ i,j : ran which_file •
    ran i.docs ∩ ran j.docs ≠ {} ⇔ i = j                    (2)
    i.title = j.title ⇔ i = j                               (3)
```

The which_file function maps the file capabilities to their particular files. The domain of this function is the set of all the capabilities for files so far created. Hence the range is the set of all the files created so far.

(1) Files cannot be empty.

(2) Documents can only be on a single file.

(3) All files can be uniqely identified by their title.

Hence a document can be uniquely identified by supplying a file title and an enclosure number.

$$\Delta THE\_FILES \triangleq [\ THE\_FILES';\ THE\_FILES\ ]$$

$$\Xi THE\_FILES \triangleq [\ \Delta THE\_FILES \mid \theta THE\_FILES' = \theta THE\_FILES\ ]$$

3.4. The Filing System

The filing system for Sercus is simply the files and documents so far created, together with the constraint that the files can only contain capabilities for documents that exist.

FILING_SYSTEM

THE_DOCS
THE_FILES

$$\forall file : ran\ which\_file \bullet$$
$$ran\ file.docs \subseteq dom\ which\_doc \qquad (1)$$

$$\forall d : which\_doc \mid ran\ file.docs \bullet$$
$$file.overall\_class \geq d.classification \qquad (2)$$

(1) Files can only contain capabilities for known documents.

(2) The overall classification of a file dominates the classifications of all the documents it contains.

$$\Delta FILING\_SYSTEM \triangleq [\ FILING\_SYSTEM'\ ;\ FILING\_SYSTEM\ ]$$

$$\Xi FILING\_SYSTEM \triangleq [\ \Delta FILING\_SYSTEM \mid \theta FILING\_SYSTEM' = \theta FILING\_SYSTEM\ ]$$

It is important to note that, in this specification, operations on documents must be performed via the filing system. This is because files can alter as a side effect of the simple operations on documents. For example, changing the classification of a document could alter the classification of any file that contains a capability for the document.

3.5 Destroying Objects

No operations to explicitly destroy objects in Sercus are defined. This is because unlike in the paper world, documents and files may be accessed by more than one user at a time. Revoking access to, or destroying, the objects then proves very difficult, as users could still possess capabilities for objects that no longer exist. As far as this specifiction is concerned, the capabilities could even potentially have been reused for other objects. Any journalled information about the object could also have been lost, and this is undesirable in a secure system. However, documents and files could be effectively removed by regrading them so that no users are cleared to see them any more.

3.6. Filing System Operations

This section describes the underlying filing system operations. These will be further defined as the specification proceeds (refer to part 2).

3.6.1 create a document

Creating a document involves providing some contents and a classification. This operation results in a new document and capability which are added to the which_doc function. Hence, the operation involves a change to the existing documents, but not to any files. In the paper world all documents are recorded on the cdr and on creation will be given a cdr number to reference them. Hence, this operation also results in the new cdr number.

create_document $_{basic\_op}$

ΔFILING_SYSTEM
ΞTHE_FILES

classification? : CLASS
contents? : $\mathbb{P}$ CAPABILITY

new_cap! : CAPABILITY
new_doc! : DOCUMENT
cdr_num! : CDR_NUM

---

contents? $\subseteq$ dom which_doc (1)
new_cap! $\notin$ dom which_doc (2)
new_doc! $\notin$ ran which_doc (3)

new_doc!.classification = classification?
new_doc!.contents = contents?
new_doc!.cdr_number = cdr_num!

$\forall$ d : ran which_doc $\bullet$ (4)
  cdr_num! $\neq$ d.cdr_number
  cdr_num! $\geq$ d.cdr_number

which_doc' = which_doc $\cup$ { new_cap! $\mapsto$ new_doc! } (5)

(1) The contents of a new document can only be capabilities to already known documents.

(2&3) The new document and its capability did not exist before the operation.

This means that the capability for the new document cannot be part of the supplied contents. Hence, since document contents are unalterable, no document can refer to itself.

(4) The new document has the supplied classification and contents and is assigned a cdr number. This cdr number is not one of those belonging to already existing documents, and is 'greater than' all the exiting numbers. However the new number is not necessarily the next in the sequence, and it is possible that there are 'holes' in the cdr.

(5) After the operation, the new capability will be mapped to the new document by the which_doc function. All other mappings are unchanged.

3.6.2 read a document

Opening a document does not alter existing documents or files. A capability for a document is supplied, and the operation results in the set of capabilities that make up its contents.

read_document $_{basic\_op}$

ΞFILING_SYSTEM

doc_cap? : CAPABILITY

contents! : ℙ CAPABILITY

---

contents! = which_doc(doc_cap?).contents

Note that at this level of specification there is no notion of checking clearances. This cannot be added until after the users have been specified (section 14).

3.6.3 finding documents

Users will be able to ask for documents by their cdr number or file title and enclosure number. This operation lists all the documents known to Sercus by their cdr number, but does not alter any documents or files.

list_cdr $_{basic\_op}$

ΞFILING_SYSTEM

cdr! : seq CDR_NUM

---

ran cdr! = { d : ran which_doc • d.cdr_number } (1)

∀ i, j : dom cdr! | i ≥ j • cdr!(i) ≥ cdr!(j) (2)

(1&2) Listing the cdr results in a sequence of all the cdr numbers for existing documents. This sequence is ordered.

The following operation looks up a cdr number and returns the associated document capability. The filing system is unaffected.

find_document $_{basic\_op}$

ΞFILING_SYSTEM

cdr_num? : CDR_NUM

doc_cap! : CAPABILITY

---

cdr_num? ∈ { d : ran which_doc • d.cdr_number } (1)
which_doc(doc_cap!).cdr_number = cdr_num?

(1) The cdr number must be assigned to an existing document, and the correct document capability is supplied.

This operation takes a file title and enclosure number and returns the required document capability. The filing system is unaffected.

find_filed_document$_{basic_op}$

```
ΞFILING_SYSTEM

title? : FTITLE
enc? : ℕ

doc_cap! : CAPABILITY

─────────────────────────────────────
  doc_cap! = file.docs( enc? )
where
  file : ran which_file | title = title?
```

This operation lists the contents of a file. The filing system is unaffected.

file_contents$_{basic_op}$

```
ΞFILING_SYSTEM

title? : FTITLE

caps! : ℙ CAPABILITY

─────────────────────────────────────
  caps! = file.docs
where
  file : ran which_file | title = title?
```

Note that further contraints will be added in part 2 to ensure that the user performing this operation is cleared to see the file contents.

3.6.4 add document to a file

Adding a document to an existing file involves supplying the file and the document capability. The enclosure number of the document on the file is returned. The overall file classification may have to increase, in order to dominate the classification of the new document. No other files or any of the documents are changed by this operation.

add_doc_to_file$_{basic_op}$

```
ΔFILING_SYSTEM
ΞTHE_DOCS

file_cap? : CAPABILITY
doc_cap? : CAPABILITY

file! : FILE
enc! : ℕ
```

```
file_cap? ∈ dom which_file
doc_cap? ∈ dom which_doc                                          (1)
doc_cap? ∉ ⋃{ file : ran which_file • ran file.docs }             (2)

file! ∉ ran which_file
file!.docs = which_file( file_cap? ).docs ^ ( doc_cap? )
file!.title = which_file( file_cap? ).title
file!.title_class = which_file( file_cap? ).title_class
file!.overall_class = which_file( file_cap? ).overall_class
                        lub
                      which_doc( doc_cap? ).classification

which_file' = which_file ⊕ { file_cap? ↦ file! }                  (3)

enc! = # file!.docs
```

(1&2) The capability supplied must be for an existing document that is not already on any file.

(3) After the operation, the supplied capability now refers to a new file (replacing the old file). This file is a copy of the original file except that the new document is added to it and the overall classification dominates the new document's classification. No other files are changed.

3.6.5 create a file

To create a file, a document, file title and title classification must be supplied. The overall classification is the document classification, which must of course dominate the title classification. The document cannot already be on any file. The which_file function will now also map the file capability to the new file. The existing documents are not changed and nor are any previous files.

create_file$_{basic_op}$

```
ΔFILING_SYSTEM
ΞTHE_DOCS

doc_cap? : CAPABILITY
title? : FTITLE
title_class? : CLASS

new_file! : FILE
new_cap! : CAPABILITY
```

```
doc_cap? ∈ dom which_doc                                          (1)
doc_cap? ∉  U{ file : ran which_file • ran file.docs }            (2)
title? ∉  { file : ran which_file • file.title }                  (3)
which_doc( doc_cap? ).classification ≥ title_class?               (4)

new_cap! ∉ dom which_file                                         (5)
new_file! ∉ ran which_file                                        (6)

new_file!.docs = ⟨ doc_cap? ⟩
new_file!.title = title?
new_file!.overall_class = which_doc( doc_cap? ).classification
new_file!.title_class = title_class?

which_file' = which_file ∪ { new_cap! ↦ new_file! }               (7)
```

(1&2) The capability supplied must be for an existing document that is not already on any file.

(3) The supplied title must be unique.

(4) The classification of the document to be put on the file (this will be the overall file classification) must dominate the given title classification.

(5&6) A new file capability and file are created, ie they did not exist before the operation.

(7) After the operation, the which_file function will map the new capability to the new file. No other files are changed.

3.6.6 regrade file

Both the title and overall classification of a file may be regraded. However, the overall classification of a file cannot be regraded lower than the title classification. In such cases, the title classification would have to be regraded first.

Changing the classification of a file involves supplying the file and a new classification. The following schemas specify regrading the title and the overall classifications of a file.

regrade_file_title$_{basic_op}$

```
ΔFILING_SYSTEM
ΞTHE_DOCS

file_cap? : CAPABILITY
title_class? : CLASS

file! : FILE
```

```
file_cap? ∈ dom which_file                                          (1)
file! ∉ ran which_file                                              (2)

file!.title_class = title_class?
file!.docs = which_file( file_cap? ).docs
file!.title = which_file( file_cap? ).title
file!.overall_class = which_file( file_cap? ).overall_class

which_file' = which_file ⊕ { file_cap? ↦ file! }                    (3)
```

(1) The file to be regraded must be known to the system.

(2) A new file is created, ie one not known before the operation. This is a copy of the file to be regraded, except that it has the new title classification.

Note that the title classification cannot be greater than the overall classification.

(3) The original file is replaced by the new file and no other files are changed.

regrade_file

```
ΔFILING_SYSTEM
ΔTHE_DOCS

file_cap? : CAPABILITY
overall_class? : CLASS

file! : FILE
```

```
file_cap? ∈ dom which_file                                          (1)
file! ∉ ran which_file                                              (2)

file!.overall_class = overall_class?
file!.docs = which_file( file_cap? ).docs
file!.title = which_file( file_cap? ).title
file!.title_class = which_file( file_cap? ).title_class

which_file' = which_file ⊕ { file_cap? ↦ file! }                    (3)
```

(1) The file to be regraded must be known to the system.

(2) A new file is created, ie one not known before the operation. This is a copy of the file to be regraded, except that it has the new overall classification.

Note that the Filing System schemas insist that the overall classification must dominate the classification of all the documents on the file, and consequently a file cannot be regraded lower than the least upper bound of all the documents classifications.

(3) The original file is replaced by the new file and no other files are changed.

Simply regrading a file does not alter any documents. However, regrading a document may result in a file classifcation being regraded, hence splitting the regrade_file schema into two parts.

$$regrade\_file_{basic\_op} \triangleq regrade\_file \wedge \Xi THE\_DOCS$$

3.6.7 regrade document

Regrading a document involves supplying a document and a new classification. If this document is on a file, the overall classification of the file may have to change as a consequence.

The operation is split into two parts. The first regrade schema is relevent when the document in question has not been filed, and the second when it has.

This regrade operation simply changes the classification of a document that is not on any file.

regrade_unfiled_document

$\Delta FILING\_SYSTEM$
ΞTHE_FILES

doc_cap? : CAPABILITY
class? : CLASS

doc! : DOCUMENT

---

doc_cap? $\in$ dom which_doc (1)
doc_cap? $\notin \bigcup \{$ file : ran which_file $\bullet$ ran file.docs $\}$ (2)

doc!.contents = which_doc(doc_cap?).contents
doc!.cdr_number = which_doc(doc_cap?).cdr_number
doc!.classification = class? (3)

which_doc' = which_doc $\oplus \{$ doc_cap? $\mapsto$ doc! $\}$ (4)

(1&2) The document to be regraded must be known, and is not on any file.

(3) A new document is created which is given identical contents and cdr number, but has the new classification.

(4) After the operation, the which_doc function maps the document capability to the new document. No other documents are changed.

This following schema defines the operation to change the classification of a filed document and indicates which file the document is on. However, it does not alter the overall classification of the file.

regrade_filed_document

ΔFILING_SYSTEM

doc_cap? : CAPABILITY
class? : CLASS

file_cap? : CAPABILITY

doc! : DOCUMENT

---

doc_cap? $\in$ dom which_doc (1)
doc_cap? $\in$ $\bigcup$ { file : ran which_file $\bullet$ ran file.docs } (2)

doc! $\notin$ ran which_doc (3)
doc!.contents = which_doc(doc_cap?).contents
doc!.cdr_number = which_doc(doc_cap?).cdr_number
doc!.classification = class?

which_doc' = which_doc $\oplus$ { doc_cap? $\mapsto$ doc! } (4)

doc_cap? $\in$ ran (which_file(file_cap?).docs) (5)

(1&2) The document to be regraded must be known, and must be on a file.

(3) A new document is created which is given identical contents and cdr number, but has the new classification.

(4) After the operation, the which_doc function maps the document capability to the new document. No other documents are changed.

(5) The file capability refers to the particular file that the document is on.

This schema specifies both regrading a filed document and the file it is on.

regrade_doc_and_file

regrade_filed_document
regrade_file

---

overall_class? = which_file(file_cap?).overall_class
lub
class? (1)

(1) The new overall classification of the file will be the least upper bound of its original classification and the new document classification.

The complete regrade operation is either the simple regrade if the document is not on a file, or regrading both the document and the file it is on.

$\text{regrade\_document}_{\text{basic\_op}} \triangleq$ regrade_doc_and_file
$\vee$ regrade_unfiled_document

4. Other Objects

Capabilities are not restricted to refer to only documents and files. It is possible for anonymous objects to exist about which the system knows very little, except their contents. The only interesting contents of an anonymous object is the set of capabilities it contains.

These anonymous objects could for example contain the characters for making up the non capability contents of a document that would exist in a real system.

ANON

$$contents : \mathbb{P}\ CAPABILITY$$

As for documents and files, a function to map anonymous capabilities to their associated object is required.

THE_ANONS

$$which\_anon : CAPABILITY \pfun ANON$$

$$dom\ which\_anon \subseteq caps\_for\_anon$$

Similarly the domain of this function gives the set of all existing capabilities for unknown objects, and the range gives all the existing unknowns.

$$\Delta THE\_ANONS \triangleq [\ THE\_ANONS' ;\ THE\_ANONS\ ]$$

$$\Xi THE\_ANONS \triangleq [\ \Delta THE\_ANONS \mid \theta THE\_ANONS' = \theta THE\_ANONS\ ]$$

> The contents information about anonymous objects will be required later when these objects are copied and stored.

The most useful operation on an anonymous object is to make a copy of it.

copy_anon

$$\Delta THE\_ANONS$$

$$original?, copy! : CAPABILITY$$
$$new\_anon! : ANON$$

$$copy! \notin dom\ which\_anon \quad (1)$$
$$new\_anon! \notin ran\ which\_anon \quad (2)$$
$$new\_anon!.contents = which\_anon(\ original?\ ).contents$$

$$which\_anon' = which\_anon \cup \{\ copy! \mapsto new\_anon!\ \}$$

(1&2) A completely new capability and object are created.

5. Displays

5.1 Introduction

A display in Sercus will consist of a set of windows. Some of the windows will be running untrusted software, although this will be monitored by the window software, which is trusted. Windows will have a set of capabilities that are available to them. The display will contain a visible representation of some of these capabilities. Windows that are monitoring untrusted software will have the correct classification of the information in the window displayed at all times. The windows that are not running any untrusted software will form the trusted path. Certain security critical operations will only be executed if they have been invoked from the trusted path [5].

The set of all possible capabilities can be partitioned into two sets. Trusted capabilities are associated with objects that can be trusted to maintain their own classification. The untrusted capabilities need to be supervised.

```
trusted_caps : ℙ CAPABILITY
untrusted_caps : ℙ CAPABILITY
─────────────────────────────────────────────────
( trusted_caps, untrusted_caps ) partition CAPABILITY
```

5.2 High Water Marks and Related Groups

Some of the objects in Sercus, such as files and documents, have their classification bound into them, and are trusted to maintain it correctly. Other objects are not classified as such. However, a mechanism is needed to monitor the classification of information these untrusted objects have access to. This is done by means of high water mark classifications.

Whenever untrusted software is accessing objects that do not protect their own classification, the worst case situation must be anticipated, and it is assumed that all these objects have access to all the information available to the software. Whenever one of the objects in such a related set has its classification increased, perhaps by opening a document, all the other objects are assumed to have access to the same information, and must be classified accordingly. These relationships are monitored by having related groups of objects and a high water mark classification associated with each group.

[GROUP]
Group objects are the mechanism for indicating the relationships between untrusted capabilities. Untrusted capabilities can be associated with a group object taken from the given set, and will be assumed to be related if they refer to the same group object. Capabilities can only belong to a single related set, and hence can only refer to a maximum of one group object. Each group object has a high water mark classification associated with it.

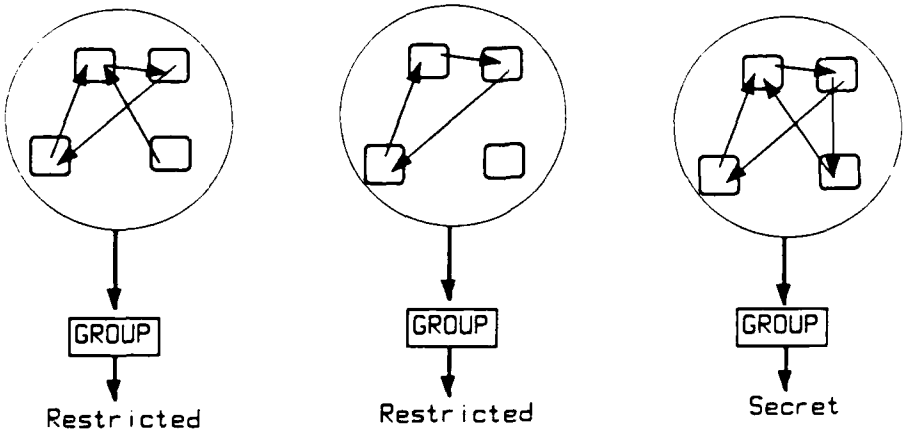


Figure 1: As far as Sercus can tell, the objects in each circle are related, and the capabilities for these objects will all be associated to the same group object. The group objects have the high water mark classification associated with them.

HIGH_WATER_MARKS

$$which\_group : CAPABILITY \pfun GROUP \quad (1)$$
$$hwm\_class : GROUP \pfun CLASS \quad (2)$$

$$dom\ which\_group \subseteq untrusted\_caps \quad (3)$$
$$ran\ which\_group \subseteq dom\ hwm\_class \quad (4)$$

(1) Capabilities are related if they refer to the same group. This is a function because each capability can only belong to a single related group, and partial because not all capabilities necessarily belong to a group.

(2) The relationship between groups and classifications is a function because each group has a single classification associated with it. This function is partial because it will be necessary to create new groups. Related sets could have the same classification, and hence the function is many to one. The domain of this function is all the group objects that have been created so far.

Note that not all the existing group objects necessarily have any capabilities associated with them.

(3) Only untrusted capabilities belong to groups.

(4) All known groups will have a classification associated with them, regardless of whether any capabilities are in the group or not.

$$\Delta HIGH\_WATER\_MARKS \triangleq [\ HIGH\_WATER\_MARKS' ; HIGH\_WATER\_MARKS\ ]$$

$$\Xi HIGH\_WATER\_MARKS \triangleq [\ \Delta HIGH\_WATER\_MARKS \mid \theta HIGH\_WATER\_MARKS' = \theta HIGH\_WATER\_MARKS\ ]$$

5.3 Windows in Sercus

A window is modelled as the set of capabilities that are available to it. The characters on the screen are uninteresting as far as this specification is concerned and are not specified.

WINDOW

$$available : \mathbb{P}\ CAPABILITY$$

[WID]

Windows need to be distinguishable even if they contain identical sets of capabilities. In order to be able to do this, windows have a unique identifier associated with them, taken from the given set.

THE_WINDOWS

$$which\_window : WID \pfun WINDOW \qquad (1)$$

The domain of the which_window function gives the set of all window identifiers that are valid, and the range all the windows that exist.

(1) This is a function because window identifiers can only refer to one window. The function is partial because not all the possible windows have yet been created, and many to one because two separate windows can contain the same set of capabilities.

$$\Delta THE\_WINDOWS \triangleq [\ THE\_WINDOWS' ; THE\_WINDOWS\ ]$$

$$\Xi THE\_WINDOWS \triangleq [\ \Delta THE\_WINDOWS \mid \theta THE\_WINDOWS' = \theta THE\_WINDOWS\ ]$$

5.4 Untrusted Software

Untrusted software has a set of capabilities that it can manipulate. All untrusted software will be monitored by a window and some of the capabilities that the untrusted software is manipulating will also be known to this window. Since it must be assumed that all untrusted capabilities available to software are related, the monitoring window will maintain a related group and high water mark classification for all those that it is aware of. This classification will be displayed in the window.

In all the following diagrams objects in the clear circles could be related and will all refer to the same group object. For clarity, these group objects have been omitted from the diagrams. Each related group has a high water mark classification associated with it. The thicker edged squares are the windows and the other squares the untrusted programs. All the capabilities in each square are those that are available to it. Any capabilities in the overlap are those that the untrusted software is manipulating that the window software is aware of.

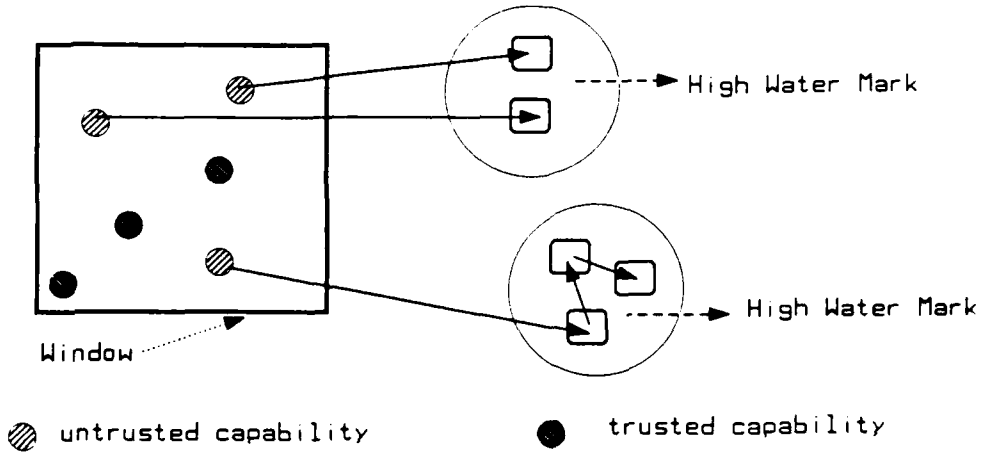


Figure 2: Window not running any untrusted software.

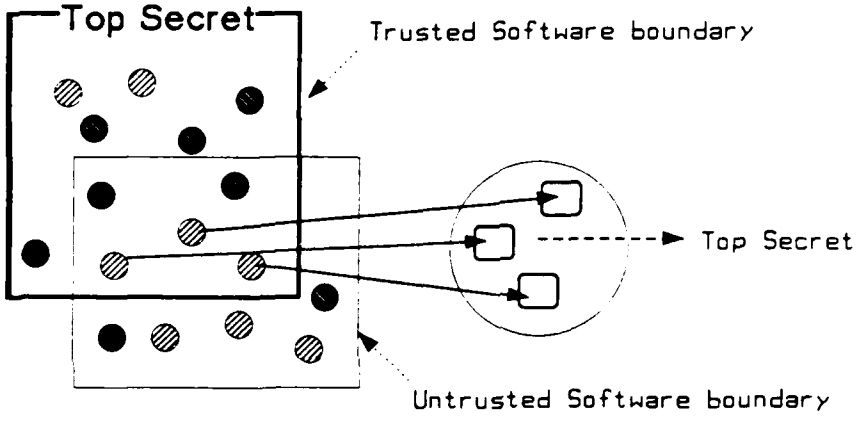


Figure 3: Window that is running untrusted software.

UNTRUSTED

using : ℙ CAPABILITY

monitoring : WID
known : ℙ CAPABILITY (1)
group : GROUP (2)
class : STR (3)

known ⊆ using ∩ untrusted_caps (4)

(1&4) All untrusted software is monitored by a window. A subset of the capabilities that the untrusted software is using will be untrusted capabilities that are known to the monitoring window.

(2&3) Untrusted software will also have a related group of capabilities associated with it and the classification for this group will be displayed.

All the known capabilities in the untrusted software will be assumed to be related and so will all belong to the same group. The monitoring window will maintain the group and high water mark

classification for this set, which may be empty. This will be fully specified in the following section.

5.5 The Display

A display consists of windows and untrusted software, together with the high water marks and related groups information.

DISPLAY

```
HIGH_WATER_MARKS
THE_WINDOWS

windows : ℙ WID
programs : ℙ UNTRUSTED
```

```
windows ⊆ dom which_window
{ p : programs • p.wid } ⊆ windows                                   (1)

∀ p, q : programs • p.monitoring = q.monitoring ⇔ p = q              (2)

∀ p : programs •
   p.known = ⋂{ p.using, untrusted_caps,
                which_window( p.monitoring ).available }             (3)
   ∀ c : p.known • which_group( c ) = p.group                        (4)
   p.class = show hwm_class( p.group )                               (5)

∀ w : windows •
   which_window( w ).available ∩ untrusted_caps
      ⊆ dom which_group                                              (6)
```

(1) The programs in the display are monitored by windows in the display. However, not all windows are monitoring untrusted programs.

(2) Windows can only monitor a single program.

(3) The known capabilities in the untrusted software are untrusted capabilities that are also available to the monitoring window (the intersection in the diagrams).

(4) All the known capabilities refer to the group object for the program, ie they could be related.

(5) The classification displayed when untrusted software is running is the correct high water mark for the related group.

(6) All the untrusted capabilities in a window will belong to some related group and hence have a high water mark classification. However, these capabilities are not necessarily related and so do not have to belong to the same group.

```
ΔDISPLAY ≙ [ DISPLAY' ; DISPLAY ]

ΞDISPLAY ≙ [ ΔDISPLAY | θDISPLAY' = θDISPLAY ]
```

5.6 Display Operations

This section defines the operations that take place in a display which are concerned with the windows and untrusted software. All the operations in Sercus will in fact be initiated from a window in a display. However this is not specified until part 2.

5.6.1 create a window

Creating a new window does not alter the high water marks or running software. A new empty window is simply added to the display.

create_window$_{basic\_op}$

ΔDISPLAY
ΞHIGH_WATER_MARKS

new_wid : WID
empty_window : WINDOW

---

new_wid $\notin$ dom which_window (1)
empty_window $\notin$ ran which_window (2)
empty_window.available = {} (3)

which_window' = which_window $\cup$ { new_wid $\mapsto$ empty_window }

windows' = windows $\cup$ { new_wid } (4)
programs' = programs (5)

(1&2) The new window and identifier are unique.

(3) The new window has no capabilities available to it initially. The window will not be running any untrusted software.

(4&5) The new window is added to the display. No other windows or any untrusted software is affected.

5.6.2 call untrusted software

Untrusted programs can only be initiated from a window in a display that is not already monitoring any untrusted software. A new group object will be created for this software. All the capabilities known by both the untrusted and the trusted software will refer to this group to indicate that they could be related. Initially the untrusted software will have no capabilities available to it and the classification will be the lowest possible.

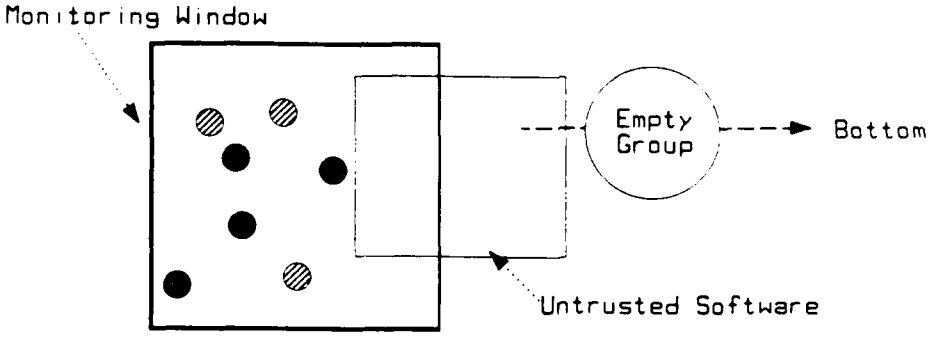


Figure 4: The window has started to run untrusted software. This software has no capabilities available to it and the classification displayed will be the lowest possible.

$run\_untrusted_{basic\_op}$

$\Delta DISPLAY$
$\Xi THE\_WINDOWS$

$wid? : WID$

$prog! : UNTRUSTED$
$group! : GROUP$

$wid? \in windows$
$wid? \notin \{ p : programs \bullet p.monitoring \}$ (1)

$group! \notin dom\ hwm\_class$ (2)
$which\_group' = which\_group$
$hwm\_class' = hwm\_class \cup \{ group! \mapsto bottom \}$

$prog! \notin programs$ (3)
$prog!.using = \{\}$
$prog!.group = group!$
$prog!.class = show\ bottom$
$prog!.monitoring = wid?$

$windows' = windows$ (4)
$programs' = programs \cup \{ prog! \}$

(1) Only an existing window that is not already monitoring untrusted software can start to run the program.

(2) A new group object is created, and added to the high water marks function with a classification of bottom. Initially there are no capabilities belonging to the group (and hence the which_group function is unchanged).

(3) A new untrusted program is created. Initially this has no capabilities available to it. The group for this software is the new group object.

(4) The new program is added into the display. No windows or existing programs are affected by the operation.

5.6.3 add to untrusted

This operation takes capabilities from a window that is running an untrusted program and makes them available to the software. This could be viewed as a 'cut and paste' operation or as passing parameters to an untrusted function. This operation does not necessarily take place between the software and the window that is monitoring it. The untrusted software could be in a completely separate window. Note that for simplicity all the following diagrams illustrate the first case.

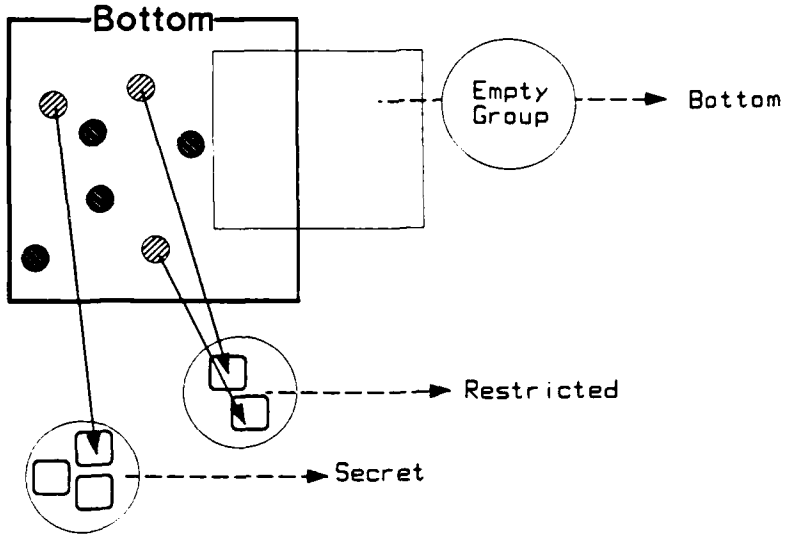


Figure 5a: The state of the window after starting to run an untrusted program and prior to passing capabilities to the untrusted software.

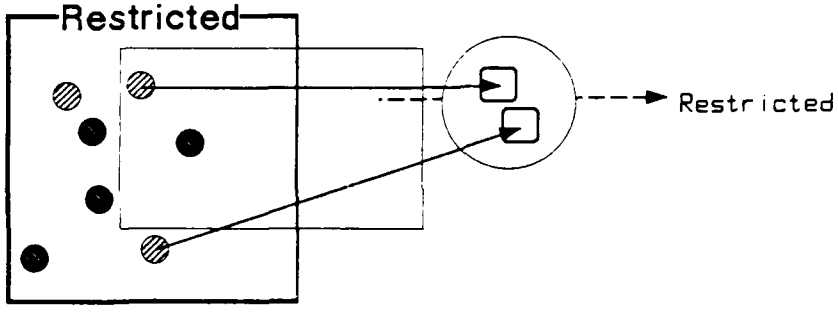


Figure 5b: The state of the window after one trusted and one untrusted capability have been added to the untrusted software.

A new group object will be created. All the untrusted capabilities being added to the software are made to refer to this group. Any capabilities that could also be related to these must be made to refer to the new group as well. However these extra capabilities may not in fact be available to the untrusted program, the worst case is always assumed.

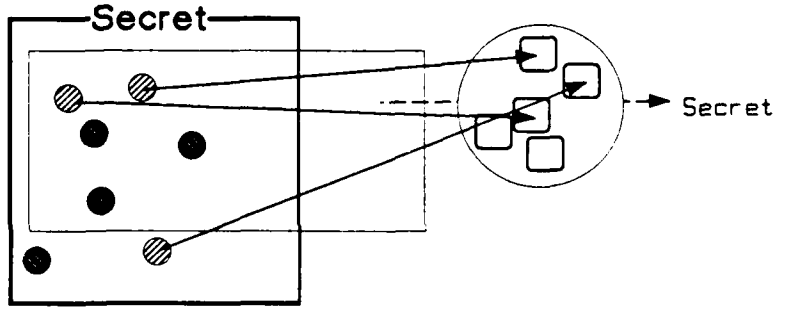


Figure 5c: After a further untrusted and two trusted capabilities have been added.

give_to_untrusted$_{basic_op}$

ΔDISPLAY
ΞTHE_WINDOWS

wid? : WID
prog? : UNTRUSTED
caps? : $\mathbb{P}$ CAPABILITY

group! : GROUP
groups! : $\mathbb{P}$ GROUP
prog! : UNTRUSTED

---

wid? $\in$ windows
prog? $\in$ programs (1)
caps? $\subseteq$ which_window(wid?).available (2)

groups! = which_group $(\!|$ caps? $|\!)$ (3)
group! $\notin$ dom hwm_class (4)

prog! $\notin$ programs (5)
prog!.using = prog?.using $\cup$ caps?
prog!.known = prog?.known $\cup$ (caps? $\cap$ untrusted_caps)
prog!.monitoring = wid?
prog!.group = group!
prog!.class = show hwm_class' (group!)

which_group'$(\!|$ dom(which_group $\rhd$ groups!) $|\!)$ = { group! } (6)
which_group' $\rhd$ groups! = which_group $\rhd$ groups! (7)

dom hwm_class' = (dom hwm_class \ groups!) $\cup$ { group! } (8)
hwm_class'(group!) = LUB { g: groups! $\bullet$ hwm_class(g) } (9)
(groups! $\cup$ { group! }) $\lhd$ hwm_class' = groups! $\lhd$ hwm_class (10)

programs' = (programs \ { prog? }) $\cup$ { prog! } (11)
windows' = windows

(1) The window and program must be in the display. However the program is not necessarily being monitored by this window.

(2) All the capabilities being given to the software are available to the window.

(3) This defines the set of related groups that all the capabilities being given to the software belong to.

(4) A new group object is created. This will give the new related group and high water mark for the untrusted software.

(5) A new untrusted program is created. This has the same capabilities available as the program the window is running plus the new capabilities. The window monitoring the untrusted does not change.

(6&7) All the untrusted capabilities being added, and those that they are related to, are made to refer to the new group. All other related sets are unchanged.

(8-10) The new group replaces all the original groups in the high water mark classifications relation. The high water mark for this new group is the least upper bound of all the classifications of the original groups.

(11) The new program replaces the original one in the display. No windows are affected.

5.6.4 take from untrusted

This operation takes capabilities from the untrusted software and adds them to a trusted window, but not necessarily the window that is running the program. This set of capabilities will be assumed to be related and will be given the high water mark classification associated with the untrusted software. This could be viewed as either a 'cut and paste' operation or as returning the result of an untrusted function.

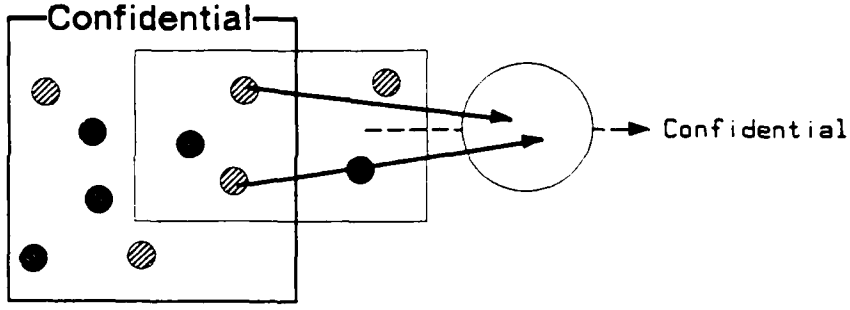


Figure 6a: The state of the window before the operation.

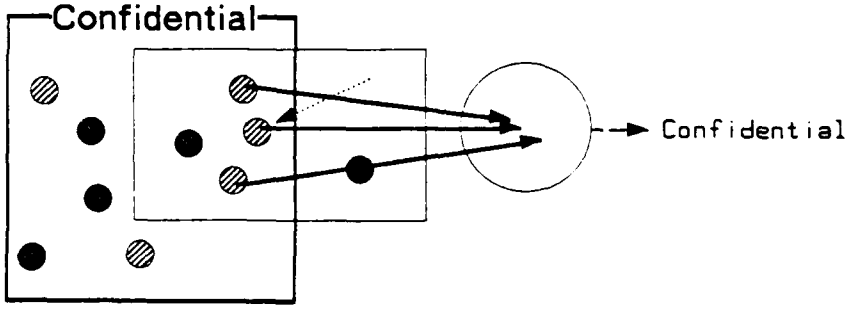


Figure 6b: After an untrusted capability has been given to the monitoring window.

take_from_untrusted$_{basic_op}$

```
ΔDISPLAY

wid? : WID
prog? : UNTRUSTED
caps? : ℙ CAPABILITY

extra! : ℙ CAPABILITY
window! : WINDOW
```

```
wid? ∈ windows                                                    (1)
prog? ∈ programs                                                  (2)
caps? ⊆ prog?.using                                               (3)

extra! = caps? ∩ untrusted_caps
         ∩ ( prog?.using \ prog?.known )                          (4)

window! ∉ ran which_window
window!.available = which_window( wid? ).available ∪ extra!
which_window' = which_window ⊕ { wid? ↦ window! }                 (5)

which_group' ( extra! ) = { prog?.group }
extra! ∉ which_group' = which_group                               (6)

hwm_class' = hwm_class
windows' = windows
programs' = programs
```

(1-3) The window and program are in the display (but the window is not necessarily monitoring the program). All the capabilities are available to the software.

(4) This defines the set of untrusted capabilities being given to the window that it was not previously aware of.

(5) A new window is created. The capabilities available to this window are those available to the initial window, plus these extra capabilities. This new window replaces the previous window.

(6) The extra capabilities now belong to the group for the untrusted software. No other groups are affected.

5.6.5 complete untrusted

This operation destroys the untrusted program that a window is monitoring. This can be viewed as returning from an untrusted function call, and will probably only be done when results have been given to the window by the 'take_from_untrusted' operation described in the previous section. No windows or high water marks are affected by this operation.

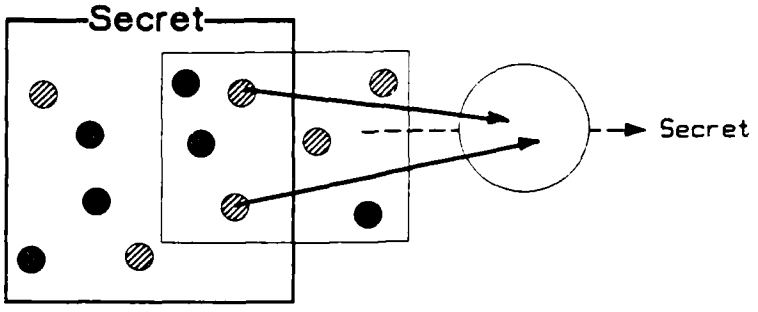


Figure 7a: The state of the window before the operation.

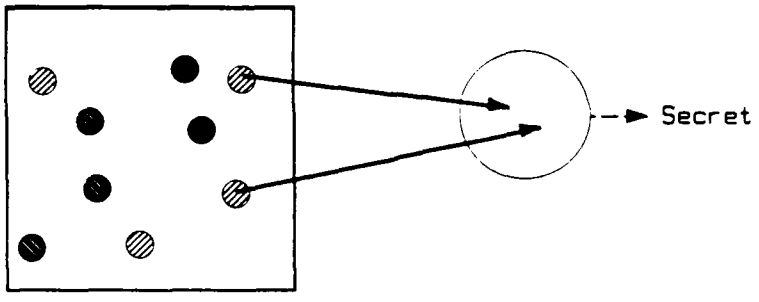


Figure 7b: After the untrusted program has finished.

complete_untrusted$_{basic_op}$

ΔDISPLAY
ΞTHE_WINDOWS
ΞHIGH_WATER_MARKS

prog? : UNTRUSTED

---

prog? $\in$ programs

programs' = programs $\setminus$ { prog? }
windows' = windows

6. Storing Capabilities

6.1 Introduction

Capabilities may be kept in a cupboard. A cupboard is simply a relationship between names and the capabilities they represent. These names are represented by printable character strings. Cupboards are similar to dictionaries or directories.

Any type of capability may be kept in the cupboard. However, when capabilities to untrusted objects are stored, they must be first copied to avoid the problems of related sets (refer to section 5.2), and the high water mark classification must be remembered as well.

6.2 Cupboards

CUPBOARD

$$name : STR \pfun CAPABILITY \quad (1)$$
$$class : CAPABILITY \pfun CLASS \quad (2)$$

$$dom\ class = (\ ran\ name\ ) \cap untrusted\_caps \quad (3)$$

(1) The relationship between names and capabilities is a function because each name can only refer to a single capability. The function is partial beacuse not all strings are used. There are no restrictions concerning naming capabilities more than once in the cupboard.

(2) The 'untrusted capabilities to classifications' relationship is a function because there is a single classification for each capability. The function is partial because only some capabilities are kept in the cupbaord.

(3) The cupboard keeps a classification for all the untrusted capabilities that are stored.

$$\Delta CUPBOARD \defeq [\ CUPBOARD' ; CUPBOARD\ ]$$

$$\Xi CUPBOARD \defeq [\ \Delta CUPBOARD \mid \theta CUPBOARD' = \theta CUPBOARD\ ]$$

6.2 Cupboard Operations

An empty cupboard is defined. This is a cupboard with no named capabilities and consequently no stored classifications either.

create_empty_cupboard

$$empty\_cupboard : CUPBOARD$$

$$empty\_cupboard.name = \{\}$$

The simple operations on a cupboard are find and keep.

Looking up a name in a cupboard will return the stored capability, and if this is

an untrusted capability the associated classification as well. The contents of the cupboard will be unaltered.

$find_{trusted}$

```
ΞCUPBOARD                          (1)

name? : STR

cap! : CAPABILITY

-------------------------------------
cap! = name( name? )               (2)
cap! ∈ trusted_caps
```

$find_{untrusted}$

```
ΞCUPBOARD                          (1)

name? : STR

cap! : CAPABILITY
class! : CLASS

-------------------------------------
cap! = name( name? )               (2)
cap! ∈ untrusted_caps
class! = class( cap! )             (3)
```

(1) The cupboard is unaffected by find.

(2) The capability returned is the the one that was stored with the given name.

(3) If the capability is untrusted, the correct classification is returned.

Keeping a capability in the cupboard adds the capability, and the classification if it is an untrusted capability, under the supplied name.

$keep_{trusted}$

```
ΔCUPBOARD

name? : STR
cap? : CAPABILITY

-------------------------------------
cap? ∈ trusted_caps
name? ∉ dom name                   (1)
name' = name ∪ { name? ↦ cap? }    (2)
```

$keep_{untrusted}$

```
ΔCUPBOARD

name? : STR
cap? : CAPABILITY
class? : CLASS

-------------------------------------
cap? ∈ untrusted_caps
name? ∉ dom name                   (1)
name' = name ∪ { name? ↦ cap? }    (2)
class' = class ∪ { cap? ↦ class? } (3)
```

(1) No capability has already been stored with the supplied name.

(2) The new name-capability relation is added to the cupboard. No other relationships are affected.

(3) If the capability being added is untrusted, its classification will also be remembered. No other classifications are affected.

Note that what is in fact kept in the cupboard will be a capability to a copy of the object. This removes the problem side effects due to operations involving related sets of capabilities, and is fully specified in part 2.

7. Users

7.1 Introduction

As in the paper world users can have specific roles. In Sercus the special kinds of users are 'system security officers' and 'registry clerks'. Only registry clerks will be able to create new files and only security officers will be able to regrade documents and files. Security officers will also be able to change users clearances and create new users.

These specific roles in Sercus will be indicated by a user having a clearance which dominates the particular classification.

```
sso   : CLASS
clerk : CLASS
```

For example, users are registry clerks if their clearance dominates the clerk classification.

7.2 The Users of Sercus

[PASSWORD]
Users of sercus will have a password, which is taken from the given set, a clearance and a cupboard in which to store capabilities.

USER

$$
\begin{array}{l}
password : PASSWORD \\
clearance : CLASS \\
cupboard : CUPBOARD
\end{array}
$$

[UID]
Users can be uniquely identified by a UID taken from the given set. At least one of the users must be a system security officer.

THE_USERS

$$
\begin{array}{ll}
which\_user : UID \pfun USER & (1) \\
my\_display : UID \pfun DISPLAY & (2) \\
\hline
dom\ my\_display \subseteq dom\ which\_user & (3) \\
\forall a, b : ran\ my\_display \bullet & \\
\quad dom\ a.hwm\_class \cap dom\ b.hwm\_class \neq \{\} \Leftrightarrow a = b & (4) \\
\quad a.windows \cap b.windows \neq \{\} \Leftrightarrow a = b & (5) \\
\forall x, y : ran\ which\_user \bullet & \\
\quad x.cupboard = y.cupboard \Leftrightarrow x = y & (6) \\
\exists sec : ran\ which\_user \bullet sec.clearance \geq sso & (7)
\end{array}
$$

The domain of the which_user function gives the set of all valid user identifiers, ie the legal users, and the range all the users. Similarly the range of the my_display function gives all the valid displays.

(1) Since the relationship between the user identifiers and the users is a function, there is a single user associated with each identifier.

The function is partial so that new users can be added to the system and one to one so that each user has a single identifier.

(2) The display will be frequently altered by the operations that users perform. Hence there is a mapping from users to their displays. The relationship is a function because each user can only have a single display, partial so that new users can log in and one to one so that each display only belongs to a single user.

(3) Not all valid users need have a display associated with them. Obviously, the users who are not currently logged in will not have a display.

(4) This predicate states that no two users can have the same group object associated with their displays. Group objects define the sets of related capabilities. This means that only a single user can have a capability for an untrusted object, and hence that users are not related.

(5&6) Users cannot share windows or cupboards.

(7) There must be at least one security officer.

$$\Delta THE\_USERS \triangleq [\ THE\_USERS' ; THE\_USERS\ ]$$

$$\Xi THE\_USERS \triangleq [\ \Delta THE\_USERS \mid \theta THE\_USERS' = \theta THE\_USERS\ ]$$

There will be a group of users who are currently logged on to Sercus. These must be valid users, ie have a user identifier associated with them.

USER_STATE

$$THE\_USERS$$
$$logged\_in : \mathbb{P}\ UID \qquad (1)$$

$$logged\_in \subseteq dom\ which\_user \qquad (2)$$

$$dom\ my\_display = logged\_in \qquad (3)$$

(1) Users can only be logged in once, and hence this is a set.

(2) Only valid users may use Sercus.

(3) All the logged in users, and only these, will have a display associated with them.

$$\Delta USER\_STATE \triangleq [\ USER\_STATE' ; USER\_STATE\ ]$$

$$\Xi USER\_STATE \triangleq [\ \Delta USER\_STATE \mid \theta USER\_STATE' = \theta USER\_STATE\ ]$$

7.3 User Operations

This section describes the basic user operations such as logging in and out. The filing system and display operations described earlier will also be performed by users, but this will be specified in part 2.

A facility to explicity delete the users of Sercus will not be provided. Instead there will be an 'authorised' classification. Users will only be able to log on if

their clearance dominates this classification. Changing a users clearance so that it is dominated by 'authorised' will therefore effectively remove them.

authorised : CLASS

7.3.1 create a new user

To create a user, a new identifier, password and clearance must be supplied. A new user with this password and an empty cupboard is added to the valid users, ie the which_user function. New users will not be logged on so will have no displays associated with them. No other users are affected.

create_user$_{basic_op}$

ΔUSER_STATE

uid? : UID
password? : PASSWORD
clearance? : CLASS

new_user! : USER
create_empty_cupboard

---

uid? $\notin$ dom which_user (1)
new_user! $\notin$ ran which_user (2)

new_user!.password = password? (3)
new_user!.clearance = clearance?
new_user!.cupboard = empty_cupboard

which_user' = which_user $\cup$ { uid? $\mapsto$ new_user! } (4)
my_display' = my_display
logged_in' = logged_in

(1&2) The user identifier and user do not already exist.

(3) The new user is given the supplied password and clearance, and an empty cupboard.

(4) The new user becomes a valid user, and no other users are affected.

Note that the ability to create new users is resticted to the system security officers, and is fully specified in part 2.

7.3.2 log in

To log onto Sercus a user identifier and password must be supplied. The user must be authorised to use Sercus, cannot already be logged on and must supply the correct password.

create_initial_display

USER_STATE
create_window$_{basic_op}$
initial_display : DISPLAY

---

initial_display $\notin$ ran my_display
initial_display.windows = { new_wid }
initial_display.programs = {}

$log\_in_{basic\_op}$

$\Delta USER\_STATE$

$uid? : UID$
$password? : PASSWORD$

$create\_initial\_display$

$uid? \notin logged\_in$ (1)
$which\_user(\ uid?\ ).password = password?$ (2)
$which\_user(\ uid?\ ).clearance \geq authorised$ (3)

$which\_window' = which\_window \cup \{\ new\_wid \mapsto empty\_window\ \}$
$my\_display' = my\_display \cup \{\ uid? \mapsto initial\_display\ \}$ (4)

$logged\_in' = logged\_in \cup \{\ uid?\ \}$ (5)
$which\_user' = which\_user$

(1) The user is not already logged on.

(2&3) The supplied password must be correct, and the user must be authorised to use the system.

(4) The display for this user consists of a new window which has no capabilities available to it. No other displays are altered.

(5) This user is now added to the set of logged on users. No other users, nor any passwords or clearances are affected.

7.3.3 log out

In order to log out, users must obviously be logged in. The user identifier and display are simply removed from the logged_in set.

$log\_out_{basic\_op}$

$\Delta USER\_STATE$

$uid? : UID$

$uid? \in logged\_in$

$my\_display' = \{\ uid?\ \} \lhd my\_display$ (1)
$which\_user' = which\_user$ (2)

$logged\_in' = logged\_in \setminus \{\ uid?\ \}$ (3)

(1) The users display is deleted.

(2) No passwords, clearances or cupboards are affected.

(3) The user is removed from the logged_in set.

7.3.4 change clearance

Users may have their clearances reviewed by the security officer. For

simplicity of implementation, clearances may only be changed when users are not logged on.

change_clearance$_{basic_op}$

```
ΔUSER_STATE

uid? : UID
clearance? : CLASS

user! : USER
```

```
uid? ∈ dom which_user
uid? ∉ logged_in                                    (1)

user!.clearance = clearance?                        (2)
user!.password = which_user( uid? ).password
user!.cupboard = which_user( uid? ).cupboard

which_user' = which_user ⊕ { uid? ↦ user! }         (3)
my_display' = my_display

logged_in' = logged_in
```

(1) Users clearances can only be changed when they are logged out.

(2) Only the particular users clearance is changed.

(3) No other users are affected.

7.3.5 change password

Users may alter their password.

change_password$_{basic_op}$

```
ΔUSER_STATE

uid? : UID
password? : PASSWORD

user! : USER
```

```
uid? ∈ dom which_user
uid? ∈ logged_in                                    (1)

user!.password = password?                          (2)
user!.clearance = which_user( uid? ).clearance
user!.cupboard = which_user( uid? ).cupboard

which_user' = which_user ⊕ { uid? ↦ user! }         (3)
my_display' = my_display

logged_in' = logged_in
```

(1) Users must be logged in to change their password.

(2) Only the user's password is changed.

(3) No other users are affected.

8. Messages

8.1 Introduction

Sercus will allow simple mail messages to be sent to users. These messages will be used by the users to request operations from the registry clerks or security officer and possibly also to send documents and files to other users.

8.2 About Messages

A message contains capabilities and an indication of which user sent the message and when. The textual part of a message is uninteresting as far as this specification is concerned.

MESSAGE

$$\begin{array}{l} message : \mathbb{P}\ CAPABILITY \\ \\ from : UID \\ time : TIME \\ \hline message \subseteq trusted\_caps \qquad (1) \end{array}$$

(1) Messages may only contain capabilities for trusted objects, such as documents and files.

Note that untrusted capabilities could be sent if the objects were copied first. However for the purposes of Sercus, messages as specified should be sufficient.

8.3 Mail

The mail system in Sercus relates messages to the recipient.

MAIL

$$mail : MESSAGE \rightarrow UID$$

$$\Delta MAIL \triangleq [\ MAIL' \ ;\ MAIL\ ]$$

$$\Xi MAIL \triangleq [\ \Delta MAIL \mid \theta MAIL' = \theta MAIL\ ]$$

8.4 Mail Operations

Sending a message involves supplying some capabilities, and the appropriate user identifiers. Both trusted and untrusted capabilities will be used to create a message, the contents of the untrusted ones making up the unspecified textual part of the message.

send$_{basic_op}$

$\Delta MAIL$

caps? : $\mathbb{P}$ CAPABILITY
me?, to? : UID

message! : MESSAGE

---

message! $\notin$ dom mail
message!.message = caps? $\cap$ trusted_caps (1)
message!.from = me?
message!.time = time_now

mail' = mail $\cup$ { message! $\mapsto$ to? } (2)

(1) A new message is created which contains only those capabilities which are trusted.

(2) This message is added to the mail system. No other messages are affected.

This operation opens the next message for the particular user. Opening a message simply makes the capabilities available.

open_next_message$_{basic_op}$

$\Delta MAIL$

message? : MESSAGE
me? : UID

caps! : $\mathbb{P}$ CAPABILITY

---

mail(message?) = me? (1)

$\forall$ m : dom (mail $\rhd$ { me? }) | m $\neq$ message? $\bullet$
message?.time $\leq$ m.time (2)

caps! = message?.message
mail' = { message? } $\lhd$ mail (3)

(1) Messages may only be opened by the user they are destined for.

(2) Messages are opened in the order in which they were sent.

In order to avoid the difficulties of two messages being sent at the same time, it is assumed that either time is defined with sufficient granularity to prevent this, or that the mailing system software will randomly choose between messages with identical times. Since the '$\leq$' operator is not fully specified it could do this random choosing.

(3) The opened message is removed from the mail system.

9. Journalling

[EVENT_TYPE]
Security critical operations are recorded in a journal. This is a sequence of events, which are taken from the given set, together with the identity of the user who caused the event to occur and when it happened.

EVENT

```
event : EVENT_TYPE
caused_by : UID
time : TIME
```

The types of event journalled can be partioned into three.

```
document_events, file_events, user_events : ℙ EVENT_TYPE
```

```
⟨ document_events, file_events, user_events ⟩
                                        partition EVENT_TYPE
```

The possible journalled events are:

```
document_created, document_opened, document_regraded,
file_created, file_regraded,
user_created, clearance_changed,                    : EVENT_TYPE
```

```
document_events =
   { document_created, document_opened,document_regraded }

file_events = { file_created, file_regraded }

user_events = { user_created, clearance_changed }
```

A journal is an ordered sequence of events.

JOURNAL

```
journal : seq EVENT
```

```
∀ i, j : dom journal | i < j •
   journal( i ).time ≤ journal( j ).time          (1)
```

(1) The journal is ordered on time.

Operations can add an entry to a journal, but never remove or replace existing entries.

ΔJOURNAL

```
JOURNAL'
JOURNAL
```

```
#journal' = #journal ∧ journal' = journal
   ∨
#journal' > #journal ∧ front( journal' ) = journal
```

```
ΞJOURNAL ≙ [ ΔJOURNAL | θJOURNAL' = θJOURNAL ]
```

9.1 Journalling the Filing System

All the documents and files in Sercus will have a journal associated with them, although this cannot be insisted upon until part 2.

JOURNAL_DOCS

$$document\_journal : CAPABILITY \rightarrowtail\!\!\!\!\!\!\bullet\ JOURNAL \qquad (1)$$

$$dom\ document\_journal \subseteq caps\_for\_docs$$

$$\forall\ j : ran\ document\_journal \bullet$$
$$\{\ e : ran\ j.journal \bullet e.event\ \} \subseteq document\_events \qquad (2)$$
$$\#j.journal \geq 1 \qquad (3)$$
$$j.journal(\ 1\ ).event = document\_created \qquad (4)$$

The document_journal function maps the document capabilities to their journals. The domain of this function will be all the capabilities for documents so far created.

(1) The relationship between documents and journals is a function so that each document has a single journal associated with it. The function is one to one because each document has a unique journal, and partial because not all the possible documents have yet been created.

(2) Only document type events can be in a document journal.

(3&4) There must be at least one event in any document journal, and the first event will be its creation.

$$\Delta JOURNAL\_DOCS \triangleq [\ JOURNAL\_DOCS'\ ;\ JOURNAL\_DOCS\ ]$$

$$\Xi JOURNAL\_DOCS \triangleq [\ \Delta JOURNAL\_DOCS \mid \theta JOURNAL\_DOCS' = \theta JOURNAL\_DOCS\ ]$$

Similarly for files:

JOURNAL_FILES

$$file\_journal : CAPABILITY \rightarrowtail\!\!\!\!\!\!\bullet\ JOURNAL$$

$$dom\ file\_journal \subseteq caps\_for\_files$$

$$\forall\ j : ran\ file\_journal \bullet$$
$$\{\ e : ran\ j.journal \bullet e.event\ \} \subseteq file\_events$$
$$\#j.journal \geq 1$$
$$j.journal(\ 1\ ).event = file\_created$$

$$\Delta JOURNAL\_FILES \triangleq [\ JOURNAL\_FILES'\ ;\ JOURNAL\_FILES\ ]$$

$$\Xi JOURNAL\_FILES \triangleq [\ \Delta JOURNAL\_FILES \mid \theta JOURNAL\_FILES' = \theta JOURNAL\_FILES\ ]$$

Adding to a document or file journal will always involve a capability to a document or file as appropriate, a user identifier and an event type.

document_op$_{journalled}$

```
ΔJOURNAL_DOCS
ΔJOURNAL                    {this defines how the journal is changed}

doc_cap? : CAPABILITY
event? : EVENT_TYPE
caused_by? : UID
```

```
document_journal( doc_cap? ) = θJOURNAL                                  (1)
document_journal' = document_journal ⊕ { doc_cap? ↦ θJOURNAL' }

journal' = journal ^ ⟨ [ EVENT | event = event?                          (2)
                                 caused_by = caused_by?
                                 time = time_now          ] ⟩
```

(1) Only the particular journal is changed.

(2) A new entry is added to the particular journal with the supplied event and user identifier. The time is set to be the current time.

Similarly for adding to a file journal:

file_op$_{journalled}$

```
ΔJOURNAL_FILES
ΔJOURNAL

file_cap? : CAPABILITY
event? : EVENT_TYPE
caused_by? : UID
```

```
file_journal( file_cap? ) = θJOURNAL
file_journal' = file_journal ⊕ { file_cap? ↦ θJOURNAL' }

journal' = journal ^ ⟨ [ EVENT | event = event?
                                 caused_by = caused_by?
                                 time = time_now          ] ⟩
```

Whenever files and documents are created a new journal will be created and added to the appropriate journalling function.

create_document_op$_{journalled}$

```
ΔJOURNAL_DOCS

new_cap? : CAPABILITY
caused_by? : UID

journal! : JOURNAL
```

```
journal! ∉ ran document_journal                                          (1)
document_journal' = document_journal ∪ { new_cap? ↦ journal! }

journal! = [ ⟨ [ EVENT | event = document_created                        (2)
                         caused_by = caused_by?
                         time = time_now          ] ⟩ ]
```

(1) A new journal is created and added to the document journal function. No other journals are affected.

(2) The new journal contains a single event. This is a 'document_created' event, with the supplied user identifier and current time.

Similarly for files:

create_file_op$_{journalled}$

```
ΔJOURNAL_FILES

new_cap? : CAPABILITY
caused_by? : UID

journal! : JOURNAL
```

```
journal! ∉ ran file_journal
file_journal' = file_journal ∪ { new_cap? ↦ journal! }

journal! = [ ( [ EVENT | event = file_created
                         caused_by = caused_by?
                         time = time_now        ] ) ]
```

9.2 Journalling the Users

There is a journal associated with each user identifier. The journal for users will record creation and changes in clearances and the user identifier of the user who performed the change. Logging in and out and changing passwords are not journalled in this example system, as it will not demonstrate anything new.

JOURNAL_USERS

```
user_journal : UID ⇸ JOURNAL                                    (1)
```

```
∀ j : ran user_journal •
   { e : ran j.journal • e.event } ⊆ user_events                (2)
   #j.journal ≥ 1                                               (3)
   j.journal( 1 ).event = user_created                          (4)
```

(1) The relationship between users and journals is a one to one function because users have a single unique journal associated with them, and partial because not all the possible users have yet been created.

(2) Only user type events can be in a user journal.

(3&4) There must be at least one event in any user journal, and the first event will be creation of the user.

```
ΔJOURNAL_USERS ≙ [ JOURNAL_USERS' ; JOURNAL_USERS ]

ΞJOURNAL_USERS ≙ [ ΔJOURNAL_USERS
                 | θJOURNAL_USERS' = θJOURNAL_USERS ]
```

As for documents and files user operations will add to the journal.

$user\_op_{journalled}$

```
ΔJOURNAL_USERS
ΔJOURNAL

user? : UID                                                    (1)
caused_by? : UID                                               (2)
event? : EVENT_TYPE

─────────────────────────────────────────────────────────────────
user_journal( user? ) = θJOURNAL
user_journal' = user_journal ⊕ { user? ↦ θJOURNAL' }

journal' = journal ^ ( [ EVENT | event = event?
                                 caused_by = caused_by?
                                 time = time_now        ] )
```

(1) This is the user that the operation is performed on.

(2) This gives the identifier of the user who performed the operation.

As for documents and files, whenever a new user is created a new journal will be added to the journalling function.

$create\_user\_op_{journalled}$

```
ΔJOURNAL_USERS

new_user?, caused_by? : UID

journal! : JOURNAL

─────────────────────────────────────────────────────────────────
journal! ∉ ran user_journal
user_journal' = user_journal ∪ { new_user? ↦ journal! }

journal! = [ ( [ EVENT | event = user_created
                         caused_by = caused_by?
                         time = time_now        ] ) ]
```

Part 2

The previous sections described the various components of Sercus independently, ie the documents and files, displays, cupboards, users, messages and journals. The remainder of the document defines how these basic operations are performed by the users and initiated from one of the windows in their display. Further constraints will be added to the basic operations as the system is built up.

Trusted capabilities are those that refer to files and documents.

```
caps_for_docs ∪ caps_for_files = trusted_caps
caps_for_anon = untrusted_caps
```

10. The Journalled Filing System

Many of the filing system operations described earlier (section 3.6) will result in a journal entry being added to the appropriate journal. To this end the journalled filing system is defined, and all the basic operations will be performed on this system rather than the filing system alone.

JOURNALLED_FILING_SYSTEM

FILING_SYSTEM
JOURNAL_DOCS
JOURNAL_FILES

---

dom document_journal = dom which_doc (1)
dom file_journal = dom which_file (2)

(1&2) All the documents and files in Sercus have a journal associated with them.

$$\Delta JOURNALLED\_FILING\_SYSTEM \triangleq [\ JOURNALLED\_FILING\_SYSTEM';\ JOURNALLED\_FILING\_SYSTEM\ ]$$

$$\Xi JOURNALLED\_FILING\_SYSTEM \triangleq [\ \Delta JOURNALLED\_FILING\_SYSTEM;\ \theta JOURNALLED\_FILING\_SYSTEM' = \theta JOURNALLED\_FILING\_SYSTEM\ ]$$

The following schemas define all the basic filing system operations to take place on the journalled system. These operations will either involve no change to any journals, or will simply add an event describing the operation to the appropriate journal.

$$create\_document_{journalled} \triangleq create\_document_{basic\_op} \gg create\_document\_op_{journalled} \wedge \Xi JOURNAL\_FILES$$

$$create\_file_{journalled} \triangleq create\_file_{basic\_op} \gg create\_file\_op_{journalled} \wedge \Xi JOURNAL\_DOCS$$

regrade_document$_{journalled}$

ΔJOURNALLED_FILING_SYSTEM
ΞJOURNAL_FILES

document_op$_{journalled}$
regrade_document$_{basic_op}$

---

event? = document_regraded

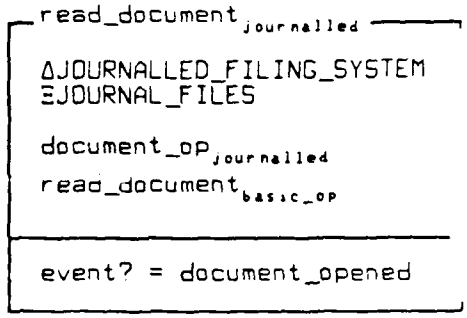
regrade_file$_{journalled}$

ΔJOURNALLED_FILING_SYSTEM
ΞJOURNAL_DOCS

file_op$_{journalled}$
regrade_file$_{basic_op}$

---

event? = file_regraded



The following filing system operations do not result in any journalled information:

$$regrade\_file\_title_{journalled} \triangleq regarde\_file\_title_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

Regrading the classification of the title of a file is not journalled only because it does not demonstrate anything new. It a larger system all changes of classification would be journalled.

$$add\_doc\_to\_file_{journalled} \triangleq add\_doc\_to\_file_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

Adding a document to a file is not journalled because all accesses to each document will still be controlled and journalled as for unfiled documents.

$$list\_cdr_{journalled} \triangleq list\_cdr_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

$$find\_document_{journalled} \triangleq find\_document_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

$$find\_filed\_document_{journalled} \triangleq find\_filed\_document_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

$$file\_contents_{journalled} \triangleq file\_contents_{basic\_op} \wedge \Xi JOURNALLED\_FILING\_SYSTEM$$

The act of aquiring a capability for an object is not journalled because having the capability does not automatically allow operations to take place. All the relevant actions will be journalled separately.

11. Journalling Users

Many of the operations performed upon the users of Sercus, such as changing clearances, need to be journalled. To this end the journalled user state is defined. The user operations defined in section 7.3 will be performed in this system rather than the basic one.

JOURNALLED_USER_STATE

USER_STATE
JOURNAL_USERS

---

dom user_journal = dom which_user (1)

(1) All the valid users have a journal associated with them.

$$\Delta JOURNALLED\_USER\_STATE \triangleq [\ JOURNALLED\_USER\_STATE'\ ;\ JOURNALLED\_USER\_STATE\ ]$$

$$\Xi JOURNALLED\_USER\_STATE \triangleq [\ \Delta JOURNALLED\_USER\_STATE \mid \theta JOURNALLED\_USER\_STATE' = \theta JOURNALLED\_USER\_STATE\ ]$$

For simplicity, only creating users and changing their clearances will be journalled.

$$create\_user_{journalled} \triangleq create\_user_{basic\_op}[\ uid? / new\_user?\ ] \wedge create\_user\_op_{journalled}$$

$change\_clearance_{journalled}$

$change\_clearance_{basic\_op}[\ uid? / user?\ ]$
$user\_op_{journalled}$

---

event? = clearance_changed

These basic operations need to have the identifier of the particular user renamed to be consistent with the names that the journalling schemas use.

$$log\_in_{journalled} \triangleq log\_in_{basic\_op} \wedge \Xi JOURNAL\_USERS$$

$$log\_out_{journalled} \triangleq log\_out_{basic\_op} \wedge \Xi JOURNAL\_USERS$$

$$change\_password_{journalled} \triangleq change\_password_{basic\_op} \wedge \Xi JOURNAL\_USERS$$

12. Promoting the Operations Involving Capabilities

Operations in Sercus will be performed by the users, or software running on their behalf, from one of the windows of their display. The filing system, cupboard and mail operations may alter the set of capabilities available to the window, but will not create or destroy windows and programs. The following schema defines this situation.

$\Delta OP_{capability}$

$\Delta DISPLAY$	{this defines how the display alters}
$window\_id? : WID$	{this identifies the particular window}
$\Delta WINDOW$	{this defines how the window alters}
$\Delta UNTRUSTED$	{this defines how untrusted software alters}

$$window\_id? \in windows \quad (1)$$
$$\theta WINDOW = which\_window(window\_id?) \quad (2)$$
$$which\_window' = which\_window \oplus \{ window\_id? \mapsto \theta WINDOW' \} \quad (3)$$
$$windows' = windows \quad (4)$$
$$window\_id? \notin \{ p : programs \bullet p.monitoring \} \Leftrightarrow programs' = programs \quad (5)$$
$$window\_id? = monitoring \Leftrightarrow programs' = programs \setminus \{ \theta UNTRUSTED \} \cup \{ \theta UNTRUSTED' \} \wedge monitoring' = monitoring \quad (6)$$

(1-3) The window in question is one of those in the display, and it is this, and only this, window that will be changed by the operation.

(4) No windows are created or destroyed in the display.

(5) If the window is not monitoring any untrusted software then none will be changed by the operation.

(6) If the window is monitoring untrusted software then it is this, and only this, software that will be altered by the operation. The window continues to monitor the software.

Security critical operations, and some operations that need to be properly controlled to prevent untrusted software exploiting signalling channels [5], must be performed on the trusted path. In a display those windows that are not running untrusted software form the trusted path. These operations will not alter any high water mark classifications or related groups of untrusted capabilities.

$TRUSTED\_PATH_{capability}$

$\Delta OP_{capability}$

$\exists HIGH\_WATER\_MARKS$ (1)

$$window\_id? \notin \{ p : programs \bullet p.monitoring \} \quad (2)$$

(1) Operations on the trusted path will not alter any high water marks or related groups of untrusted capabilities.

(2) Trusted path windows are only those that are not monitoring any untrusted software.

Operations are performed by users in one of the windows of their display. The password, clearance and cupboard will be unaltered by the filing system and mail operations. No other users will be directly affected by these operations that a user can perform.

$\Phi OP_{filing\_system USER\_STATE}$

$\Delta OP_{capability}$
ΔJOURNALLED_USER_STATE
ΞJOURNAL_USERS

caused_by? : UID **{this is the user performing the operation}**

---

my_display(caused_by?) = θDISPLAY
my_display' = my_display $\oplus$ { caused_by? $\mapsto$ θDISPLAY' } (1)

which_user' = which_user (2)
logged_in' = logged_in

(1) The display for the particular user will be altered by operations. *No other users displays will be changed.*

(2) The password, clearance and cupboard remain the same.

Note that no journalling information about the users is added. This is because each document and file maintains its own journal of accesses and regrades.

Some of the operations in Sercus must be performed by specific users, such as security officers or registry clerks.

SSO $\triangleq$ [$\Phi OP_{filing\_system USER\_STATE}$
| which_user(caused_by?).clearance $\geq$ sso]

CLERK $\triangleq$ [$\Phi OP_{filing\_system USER\_STATE}$
| which_user(caused_by?).clearance $\geq$ clerk]

The operations upon the cupboard do not alter the display except by adding capabilities to the window that the operation is performed from. These operations are performed by a user, and will not alter the password or clearance.

$\Phi OP_{cupboard USER\_STATE}$

$\Delta OP_{capability}$
$\Delta USER\_STATE$
$\Delta USER$
$\Delta CUPBOARD$ {this defines how the cupboard changes}

caused_by? : UID

---

my_display(caused_by?) = θDISPLAY
my_display' = my_display ⊕ { caused_by? ↦ θDISPLAY' } (1)

which_user(caused_by?) = θUSER
which_user' = which_user ⊕ { caused_by? ↦ θUSER' } (2)

cupboard = θCUPBOARD (3)
cupboard' = θCUPBOARD'
password' = password
clearance' = clearance

logged_in' = logged_in

(1) The display for the particular user will be altered by operations, and no others.

(2&3) The cupboard is altered by the operation, but the password and clearance remain the same.

13. The Operations upon the Filing System

13.1 Filing System Operations in a Display

The journalled filing system operations described in section 10 will be performed in a particular window and display, and may alter the contents of the window.

The only security critical operations on the filing system are regrading documents and files. However, it would be inconvenient if Trojan Horses in untrusted software created files or documents and added documents to files in a random manner. In fact if other untrusted software cooperated, this could form a significant signalling channel. To prevent this, these operations can only be performed from the trusted path.

$regrade\_document_{TP\_FILING\_SYSTEM}$	
$TRUSTED\_PATH_{capability}$ $regrade\_document_{journalled}$	
doc_cap? ∈ available	(1)
θWINDOW' = θWINDOW	(2)

(1) This is the basic regrade operation, together with the constraint that the document being regraded is available to the window.

(2) The window is unaltered by the operation.

$regrade\_file_{TP\_FILING\_SYSTEM}$	
$TRUSTED\_PATH_{capability}$ $regrade\_file_{journalled}$	
file_cap? ∈ available	(1)
θWINDOW' = θWINDOW	(2)

$regrade\_file\_title_{TP\_FILING\_SYSTEM}$	
$TRUSTED\_PATH_{capability}$ $regrade\_file\_title_{journalled}$	
file_cap? ∈ available	(1)
θWINDOW' = θWINDOW	(2)

(1) This is the basic regrade operation, together with the constraint that the file being regraded is available to the window.

(2) The window is unaltered by the operation.

$create\_file_{TP\_FILING\_SYSTEM}$	
$TRUSTED\_PATH_{capability}$ $create\_file_{journalled}$	
doc_cap? ∈ available	(1)
available' = available ∪ new_cap!	(2)

(1) This is the basic operation for creating files, together with the constraint that the document capability is available to the window.

(2) After the operation the new capability is available to the window.

add_doc_to_file$_{TP_FILING_SYSTEM}$

TRUSTED_PATH$_{capability}$
add_doc_to_file$_{journalled}$

{ doc_cap?, file_cap? } ⊆ available (1)
θWINDOW' = θWINDOW (2)

(1) This is the basic operation, together with the constraint that both the file and document capabilities are available to the window.

(2) The window is unaltered by the operation.

Creating a document involves supplying both trusted and untrusted capabilities. The trusted capabilities will be for other documents and will make up the capability contents of the new document. The untrusted capabilities will be for other objects which will go towards making up the textual contents of the document. The only important part of the untrusted capabilities as far as this specification is concerned is the high water mark classification.

create_document$_{TP_FILING_SYSTEM}$

TRUSTED_PATH$_{capability}$
create_document$_{journalled}$

caps? : ℙ CAPABILITY

---

caps? ⊆ available (1)
contents? = caps? ∩ trusted_caps (2)
classification? ≥ LUB{ c : caps? ∩ untrusted_caps •
hwm_class which_group(c) } (3)

available' = available ∪ { new_cap! } (4)

(1) All the supplied capabilities are available to the window.

(2) The contents capabilities for the basic operation are the supplied trusted capabilities.

(3) The classification given to the document must dominate all the high water mark classifications of the untrusted capabilities that go towards the document contents.

(4) After the operation the new capability is available to the window.

Reading a document, looking up documents in the cdr or filelist and listing the contents of a file do not have to be performed from the trusted path, although they could be. These operations will make further capabilties available to the window. Reading documents and listing the contents of a file will only be permitted if the user is cleared to do so. However this cannot be described until the section 13.2 when users have been further specified.

find_document$_{FILING_SYSTEM}$

$\Delta OP_{capability}$
$\exists HIGH\_WATER\_MARKS$ (1)

$find\_document_{journalled}$

---

window_id? = monitoring ⇔ (2)
  using' = using ∪ { doc_cap! }
  class' = class
  group' = group
  θWINDOW' = θWINDOW

window_id? ≠ monitoring ⇔ (3)
  θUNTRUSTED' = θUNTRUSTED
  available' = available ∪ { doc_cap! }

find_filed_document$_{FILING_SYSTEM}$

$\Delta OP_{capability}$
$\exists HIGH\_WATER\_MARKS$ (1)

$find\_filed\_document_{journalled}$

---

window_id? = monitoring ⇔ (2)
  using' = using ∪ { doc_cap! }
  class' = class
  group' = group
  θWINDOW' = θWINDOW

window_id? ≠ monitoring ⇔ (3)
  θUNTRUSTED' = θUNTRUSTED
  available' = available ∪ { doc_cap! }

file_contents$_{FILING_SYSTEM}$

$\Delta OP_{capability}$
$\exists HIGH\_WATER\_MARKS$ (1)

$file\_contents_{journalled}$

---

window_id? = monitoring ⇔ (2)
  using' = using ∪ caps!
  class' = class
  group' = group
  θWINDOW' = θWINDOW

window_id? ≠ monitoring ⇔ (3)
  θUNTRUSTED' = θUNTRUSTED
  available' = available ∪ caps!

(1) Capabilities for documents are trusted and therefore no high water marks need be changed as a result of this operation.

(2) If the operation is performed from untrusted software, the capabilities available to that software are simply increased by the new capability (or capabilities). The monitoring window is unaffected.

(3) If the operation was performed in a trusted path window the available capabilities are simply increased by the new capability (or capabilities) and any untrusted software in the display is unaffected.

$list\_cdr_{FILING\_SYSTEM}$

$\Delta OP_{capability}$
$list\_cdr_{journalled}$

$\theta DISPLAY' = \theta DISPLAY$ (1)

(1) Because windows and untrusted software are modelled by the set of capabilities available to them, listing the cdr does alter the display.

Opening a document will reveal further (trusted) capabilities. There are no restrictions as to where this operation can be performed as documents will only be able to be read if the users clearance dominates the classification (this will be specified later after users have been introduced, see section 14).

Reading a document from trusted software simply makes the contents capabilities available to the window.

$read\_document_{TRUSTED\_PATH}$

$\Delta OP_{capability}$
$read\_document_{journalled}$

$window\_id? \notin \{ p : programs \bullet p.monitoring \}$ (1)
$doc\_cap? \in available$ (2)

$available' = available \cup contents!$ (3)
$\theta UNTRUSTED' = \theta UNTRUSTED$
$\theta HIGH\_WATER\_MARKS' = \theta HIGH\_WATER\_MARKS$

(1&2) The window is not running any untrusted software, and the document capability is available to this window.

(3) The capabilities available to the window are simply increased by the contents of the document. No untrusted software or high water marks are affected.

Whenever a document is opened by untrusted software, the high water mark may have to be increased to take account of the classification of the contents. The following diagrams illustrate this.

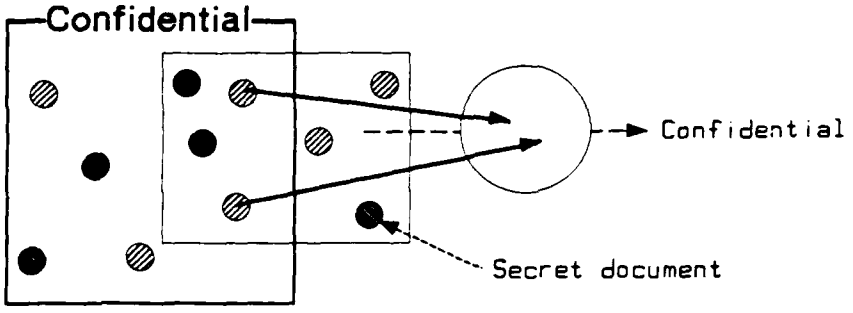


Figure 8a: The state of a window running untrusted software before opening a document.

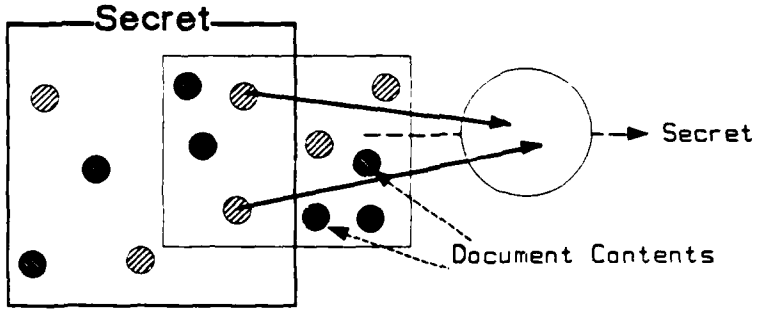


Figure 8b: After opening the document to reveal secret information.

$read\_document_{UNTRUSTED}$

$\Delta OP_{capability}$
$read\_document_{journalled}$

```
window_id? = monitoring                                    (1)
doc_cap? ∈ using                                           (2)

using' = using ∪ contents!                                 (3)
known' = known                                             (4)
group' = group
class' = show hwm_class'( group )
hwm_class' = hwm_class ⊕                                   (5)
                { group ↦ which_doc( doc_cap? ).classification
                                lub
                          hwm_class( group )
                }
θWINDOW' = θWINDOW
```

(1&2) The window is running the untrusted software, and the document capability is available to the untrusted program.

(3) The capabilities available to the untrusted software are increased by the contents of the document.

(4) No capabilities are added to the monitoring window.

(5) The high water mark for the untrusted software will be the least upper bound of the initial classification and that for the document.

The complete operation is specified to be either of the above:

$$read\_document_{FILING\_SYSTEM} \triangleq read\_document_{TRUSTED\_PATH} \vee read\_document_{UNTRUSTED}$$

13.2 Filing System Operations Performed by Users

The filing system operations are performed by users in one of the windows of their display, and do not alter cupboards, passwords or clearances. Certain of these operations must be performed by specific users.

$$regrade\_document_{USER} \triangleq regrade\_document_{TP\_FILING\_SYSTEM} \wedge SSO$$

$$regrade\_file_{USER} \triangleq regrade\_file_{TP\_FILING\_SYSTEM} \wedge SSO$$

$$regrade\_file\_title_{USER} \triangleq regrade\_file\_title_{TP\_FILING\_SYSTEM} \wedge SSO$$

$$create\_document_{USER} \triangleq create\_document_{TP\_FILING\_SYSTEM} \wedge \Phi OP_{filing\_systemUSER\_STATE}$$

$$create\_file_{USER} \triangleq create\_file_{TP\_FILING\_SYSTEM} \wedge CLERK$$

$$add\_doc\_to\_file_{USER} \triangleq add\_doc\_to\_file_{TP\_FILING\_SYSTEM} \wedge \Phi OP_{filing\_systemUSER\_STATE}$$

$$find\_document_{USER} \triangleq find\_document_{FILING\_SYSTEM} \wedge \Phi OP_{filing\_systemUSER\_STATE}$$

$$find\_filed\_document_{USER} \triangleq find\_filed\_document_{FILING\_SYSTEM} \wedge \Phi OP_{filing\_systemUSER\_STATE}$$

$$list\_cdr_{USER} \triangleq list\_cdr_{FILING\_SYSTEM} \wedge \Phi OP_{filing\_systemUSER\_STATE}$$

$read\_document_{USER}$

$\Phi OP_{filing\_systemUSER\_STATE}$
$read\_document_{FILING\_SYSTEM}$

---

which_user(caused_by?).clearance ≥
which_doc(doc_cap?).classification (1)

(1) In order to read a document, the users clearance must dominate the classification of the document.

$file\_contents_{USER}$

$\phi OP_{filing\_system USER\_STATE}$
$file\_contents_{FILING\_SYSTEM}$

```
  which_user( caused_by? ).clearance ≥ file.classification  (1)
where
  file : ran which_file | title = title?
```

(1) In order to list the contents of a file, the users clearance must dominate the file classification.

14. The Operations upon Cupboards

14.1 Cupboard Operations in a Display

The basic operations on a cupboard, see section 6.3, will be performed in a display.

The actions of the find operation depend on whether the operation is invoked from trusted or untrusted software and the trustworthiness of the particular capability. There are four cases, which are illustrated and specified below.

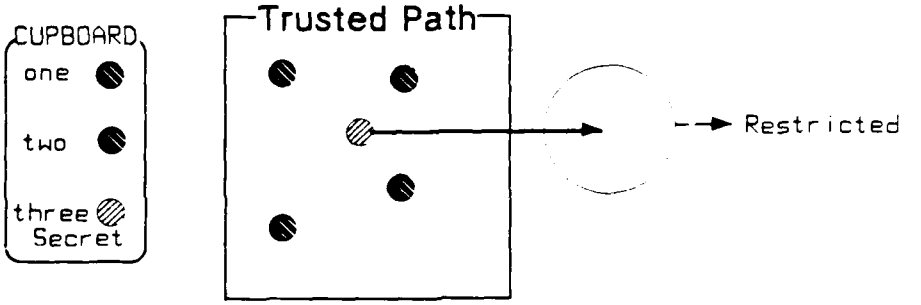


Figure 9a: A trusted path window prior to looking up an untrusted capability in the cupboard.

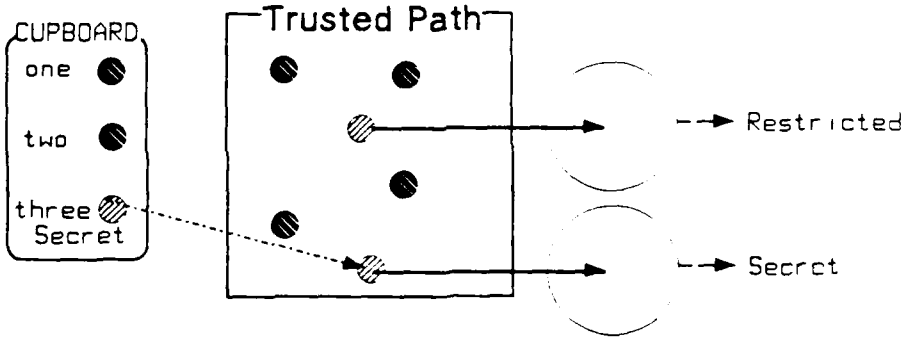


Figure 9b: Trusted path window after adding an untrusted capability from the cupboard.

$find_{utp}$

$TRUSTED\_PATH_{capability}$

$find_{untrusted}$

$$which\_group' = which\_group \cup \{ cap! \mapsto new\_group \} \quad (1)$$
$$hwm\_class' = hwm\_class \cup \{ new\_group \mapsto class! \}$$
$$available' = available \cup \{ cap! \}$$

where

$$new\_group : GROUP \mid new\_group \notin dom\ hwm\_class$$

(1) A new related group object is created for this capability and given the high water mark classification taken from the cupboard. The new capability is made available to the window. No untrusted software or other windows are affected.

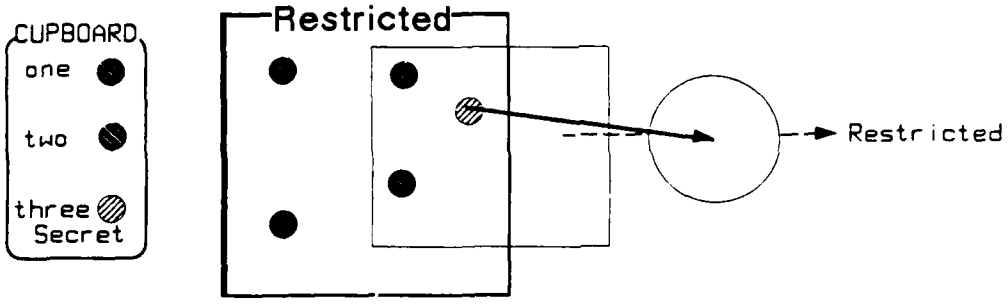


Figure 10a: The situation priior to untrusted software looking up an untrusted capability in the cupboard.

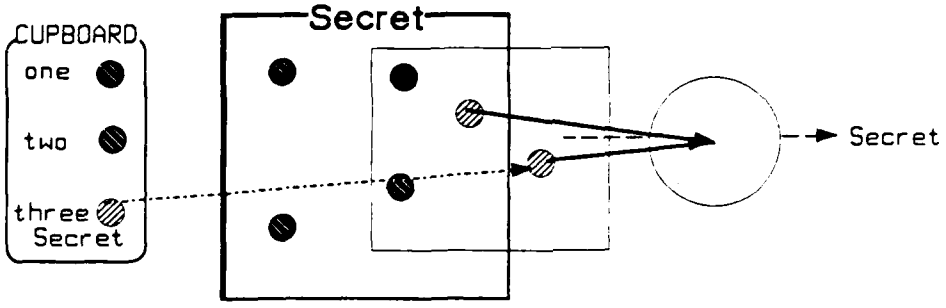


Figure 10b: After adding an untrusted capability from the cupboard.

$find_u$

$\Delta OP_{capability}$
$find_{untrusted}$

$\theta WINDOW' = \theta WINDOW$
$using' = using \cup \{ cap! \}$ (1)
$known' = known$
$group' = group$
$class' = show\ hwm\_class'(\ group\ )$
$which\_group' = which\_group \cup \{ cap! \mapsto group \}$ (2)
$hwm\_class' = hwm\_class \oplus \{ group \mapsto (\ hwm\_class(\ group\ ) \sqcup class!\ ) \}$

(1) The new capability is added to the set that the untrusted software is using. The monitoring window is unaffected.

(2) The new capability is added to the related group for that software, and the high water mark is increased as appropriate. No other high water marks or groups are affected.

Whenever trusted capabilities are taken from the cupboard no high water marks or related groups need be altered.

$find_{ttp}$

$TRUSTED\_PATH_{capability}$
$find_{trusted}$

---

available' = available ∪ { cap! } (1)

(1) The capability from the cupboard is made available to the window, and no high water marks or untrusted software change.

$find_t$

$\Delta OP_{capability}$
$find_{trusted}$

θHIGH_WATER_MARKS' = θHIGH_WATER_MARKS
using' = using ∪ { cap! } (1)
known' = known
group' = group
class' = class
θWINDOW' = θWINDOW

(1) The capability from the cupboard is added to those available to the software. The monitoring window and high water marks are unaffected.

The complete find operation is one of the four cases.

$$find_{DISPLAY} \triangleq find_{utp} \vee find_{ttp} \vee find_u \vee find_t$$

It would be a signalling channel if untrusted software could store capabilities in the cupboard, so this operation must be performed from the trusted path. No untrusted software will be affected by storing capabilities, except in that the capability can now be found by software with access to the same cupboard.

$keep_u$

$TRUSTED\_PATH_{capability}$
copy_anon (1)
$keep_{untrusted}$

---

original? ∈ available ∩ untrusted_caps (2)
cap? = copy!
class? = hwm_class(which_group(original?)) (3)
θWINDOW' = θWINDOW

(1) In order to remove the problem of the related sets (see section 5.2) whenever untrusted capabilities are stored, what is in fact stored is a capability to a new object with a copy of the original's contents.

(2) The capability being kept must be one of those available to the window.

(3) The classification that is kept with untrusted capabilities must be correct.

$keep_t$

$TRUSTED\_PATH_{capability}$

$keep_{trusted}$

$cap? \in available \cap trusted\_caps$ (1)

$\theta WINDOW' = \theta WINDOW$

(1) The capability being kept must be one of those available to the window.

The complete operation is either case.

$$keep_{DISPLAY} \triangleq keep_u \vee keep_t$$

18. Cupboard Operations Performed by Users

The cupboard operations will be performed by any of the users of Sercus in one of the windows of their display.

$$find_{USER} \triangleq find_{DISPLAY} \wedge \Phi OP_{cupboardUSER\_STATE}$$

$$keep_{USER} \triangleq keep_{DISPLAY} \wedge \Phi OP_{cupboardUSER\_STATE}$$

15. The Operations upon the Mail System

15.1 Mail Operations Performed in a Display

The mail operations will be performed in a window of a display. No high water marks or untrusted software are altered. In order to prevent untrusted software using messages as a signalling channel, the message operations must be performed from the trusted path.

$send\_message_{DISPLAY}$

$TRUSTED\_PATH_{capability}$
$send_{basic\_op}$

caps? ⊆ available (1)
θWINDOW' = θWINDOW (2)

(1) The capabilities making up the message must all be available to the window.

(2) The window is not altered by the operation.

$open\_next\_message_{DISPLAY}$

$TRUSTED\_PATH_{capability}$
$open\_next\_message_{basic\_op}$

available' = available ∪ caps! (1)

(1) Opening a message will make its capabilities available to the window. These capabilities are all trusted, and therefore no high water marks or related groups need change.

15.2 Mail Operations Performed by Users

The mail operations may be performed by any user of Sercus. As for the filing system operations, only the capabilities available to the display of the particular user may alter. The password, clearance and cupboard cannot change.

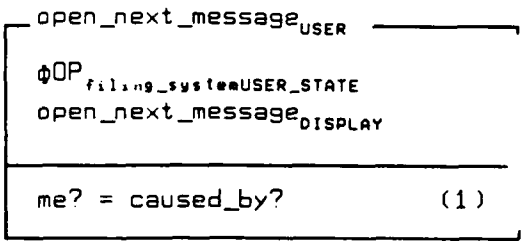
$send\_message_{USER}$

$\phi OP_{filing\_systemUSER\_STATE}$
$send\_message_{DISPLAY}$

me? = caused_by? (1)
to? ∈ dom which_user (2)

(1) The user identifier for the sender of the message will be correct.

(2) Messages may only be sent to valid users.



(1) A message may only be opened if the message is destined for the owner of the window.

16. Promoting the Operations Involving a Display

The operations that alter the windows and software in a display will also be initiated from one of the windows in the display. The following schemas define this situation.

$\Delta OP_{display}$

```
ΔDISPLAY

window_id? : WID
ΔWINDOW
ΔUNTRUSTED
```

```
window_id? ∈ windows
θWINDOW = which_window( window_id? )
θWINDOW' = which_window'( window_id? )
```

As for the $\Delta OP_{capability}$ schema the particular display and window that the operation is performed from are defined. This schema does not define how the untrusted software and windows are altered as this is defined in the basic operation schemas.

$TRUSTED\_PATH_{display}$

```
ΔOP_display
ΞHIGH_WATER_MARKS
```

```
window_id? ∉ { p : programs • p.monitoring }
```

The operations upon a display are performed by a user. Users may only alter their own display, and no others. No passwords, clearances or cupboards will be affected by the operations upon the display.

$\Phi OP_{displayUSER\_STATE}$

```
ΔOP_display
ΔJOURNALLED_USER_STATE
ΞJOURNAL_USERS

caused_by? : UID
```

```
my_display( caused_by? ) = θDISPLAY
my_display' = my_display ⊕ { caused_by? ↦ θDISPLAY' }    (1)

which_user' = which_user                                  (2)
logged_in' = logged_in                                    (3)
```

(1) The display for the particular user will be altered by operations, and no others.

(2&3) Operations upon the display do not alter the password, clearance or cupboard of any users, nor the set of logged in users.

17. The Operations upon the Display of a User

17.1 Display Operations performed from a Display

The operations on a display described in section 5.6 will be initiated from one of the windows of a display. These schemas combine the basic operation with the schema that defines operations to take place in a display, and ties the components together.

create_window$_{DISPLAY}$

$TRUSTED\_PATH_{display}$
$create\_window_{basic\_op}$

---

empty_window = θWINDOW'

run_untrusted$_{DISPLAY}$

$TRUSTED\_PATH_{display}$
$run\_untrusted_{basic\_op}$

---

window_id? = wid? (1)
θUNTRUSTED' = prog!

complete_untrusted$_{DISPLAY}$

$\Delta OP_{display}$
$complete\_untrusted_{basic\_op}$

---

window_id? = prog?.monitoring (1)
θUNTRUSTED = prog?

give_to_untrusted$_{DISPLAY}$

$\Delta OP_{display}$
$give\_to\_untrusted_{basic\_op}$

---

window_id? = wid? (1)
θUNTRUSTED = prog?
θUNTRUSTED' = prog!

take_from_untrusted$_{DISPLAY}$

$\Delta OP_{display}$
$take\_from\_untrusted_{basic\_op}$

---

window_id? = wid? (1)
θUNTRUSTED = prog?
θWINDOW' = window'

(1) This ties up the window performing the operations with the window from the basic operation schema.

17.2 Display Operations performed by Users

Users may perform the operations to alter their display, such as creating windows and running untrusted software. These operations are not restricted to users with certain roles, however a user can only affect their own display.

$give\_to\_untrusted_{USER} \triangleq give\_to\_untrusted_{DISPLAY} \wedge \Phi OP_{displayUSER\_STATE}$

$take\_from\_untrusted_{USER} \triangleq take\_from\_untrusted_{DISPLAY} \wedge \Phi OP_{displayUSER\_STATE}$

$create\_window_{USER} \triangleq create\_window_{DISPLAY} \wedge \Phi OP_{displayUSER\_STATE}$

$run\_untrusted_{USER} \triangleq run\_untrusted_{DISPLAY} \wedge \Phi OP_{displayUSER\_STATE}$

$complete\_untrusted_{USER} \triangleq complete\_untrusted_{DISPLAY} \wedge \Phi OP_{displayUSER\_STATE}$

18. The Operations upon Users

18.1 User Operations performed in a Display

With the exception of logging in, the operations on users will be requested from one of the windows of a display. All these operations must be performed from the trusted path. No capabilities are involved in these operations and hence there will be no change to the capabilities available to the window.

$$create\_user_{DISPLAY} \triangleq create\_user_{journalled} \wedge TRUSTED\_PATH_{display} \wedge EWINDOW$$

$$change\_password_{DISPLAY} \triangleq change\_password_{journalled} \wedge TRUSTED\_PATH_{display} \wedge EWINDOW$$

$$change\_clearance_{DISPLAY} \triangleq change\_clearance_{journalled} \wedge TRUSTED\_PATH_{display} \wedge EWINDOW$$

$$log\_out_{DISPLAY} \triangleq log\_out_{journalled} \wedge TRUSTED\_PATH_{display} \wedge EWINDOW$$

18.2 Operations upon Users performed by the Users

The operations on the users of Sercus will be performed by the users in one of the windows of their display. Creating users and changing clearances must be performed by a system security officer.

$$create\_user_{USER} \triangleq create\_user_{DISPLAY} \wedge SSO$$

$$change\_clearance_{USER} \triangleq change\_clearance_{DISPLAY} \wedge SSO$$

log_out_{USER}

$\Phi OP_{displayUSER\_STATE}$ $log\_out_{DISPLAY}$
uid? = caused_by? (1)

$change\_password_{USER}$

$\Phi OP_{displayUSER\_STATE}$ $change\_password_{DISPLAY}$
uid? = caused_by? (1)

(1) Users may only change their own password and log themselves out.

Note that logging in is not specified as unlike all the other operations, it is not performed by a logged in user in one of the windows of their display.

19. Sercus - the Complete System

The complete system for Sercus is the journalled filing system, journalled users, the mail system and anonymous objects, together with constraints to tie them together. This section puts all the components specified in the preceeding sections together, and defines the operations to take place in this state.

SERCUS

```
JOURNALLED_FILING_SYSTEM
THE_ANONS
JOURNALLED_USER_STATE
MAIL
```

```
  ∀ u : ran which_user •                                              (1)
     ran( u.cupboard.name ) ⊆ exist
     { w : which_window( u.windows ) • w.available } ⊆ exist
     { p : u.programs • p.using } ⊆ exist
where
  exist = dom which_doc ∪ dom which_file ∪ dom which_anon

{ m : ran mail • m.message } ⊆ dom which_doc                          (2)

ran mail ⊆ dom which_user                                             (3)

{ m : dom mail • m.from } ⊆ dom which_user                            (4)
```

(1) All the trusted capabilities in the cupboards, windows and software must be for existing documents and files, and untrusted capabilities must be for existing anonymous objects.

(2) Messages can only contain capabilities for existing documents.

(3) Messages may only be sent to legal users.

(4) Only legal users may send messages.

```
ΔSERCUS ≙ [ SERCUS' ; SERCUS ]

ΞSERCUS ≙ [ ΔSERCUS | θSERCUS' = θSERCUS ]
```

19.1 Initial State

The initial state of Sercus is a single user, who will be a system security officer. There will hence be a single journal on the user state. There will be no documents or files, and consequently no journals for other objects, no mail messages or anyone logged_in.

INITIAL_STATE

ΔSERCUS

---

which_doc' = {}
which_file' = {}
which_anon' = {}
mail' = {}
logged_in' = {}

dom which_user' = 1
$\forall$ u : ran which_user' • u.cupboard = {}
u.clearance $\geq$ sso
journal_users' = 1
dom journal_users' = dom which_user'

19.2 Operations on the Complete State

Filing system operations:

regrade_document $\triangleq$ regrade_document$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

regrade_file $\triangleq$ regrade_file$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

regrade_file_title $\triangleq$ regrade_file_title$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

create_document $\triangleq$ create_document$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

create_file $\triangleq$ create_file$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

add_doc_to_file $\triangleq$ add_doc_to_file$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

find_document $\triangleq$ find_document$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

find_filed_document $\triangleq$ find_filed_document$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

list_cdr $\triangleq$ list_cdr$_{User}$ $\wedge$ ESERCUS

file_contents $\triangleq$ file_contents$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

read_document $\triangleq$ read_document$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS

find $\triangleq$ find$_{USER}$ $\wedge$ EMAIL $\wedge$ EJOURNALLED_FILING_SYSTEM

keep $\triangleq$ keep$_{USER}$ $\wedge$ EMAIL $\wedge$ EJOURNALLED_FILING_SYSTEM

Mail operations:

send_message $\triangleq$ send_message$_{USER}$ $\wedge$ ETHE_ANONS
$\wedge$ EJOURNALLED_FILING_SYSTEM

open_next_message $\triangleq$ open_next_message$_{USER}$ $\wedge$ ETHE_ANONS
$\wedge$ EJOURNALLED_FILING_SYSTEM

Display operations:

create_window $\triangleq$ create_window$_{USER}$ $\wedge$ EMAIL $\wedge$ ETHE_ANONS
$\wedge$ EJOURNALLED_FILING_SYSTEM

$$run\_untrusted \triangleq run\_untrusted_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

$$give\_to\_untrusted \triangleq give\_to\_untrusted_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

$$take\_from\_untrusted \triangleq take\_from\_untrusted_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

$$complete\_untrusted \triangleq complete\_untrusted_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

User operations:

$$create\_user \triangleq create\_user_{USER} \wedge \exists MAIL \wedge \exists JOURNALLED\_FILING\_SYSTEM \wedge \exists THE\_ANONS$$

$$log\_out \triangleq log\_out_{USER} \wedge \exists MAIL \wedge \exists JOURNALLED\_FILING\_SYSTEM \wedge \exists THE\_ANONS$$

$$change\_clearance \triangleq change\_clearance_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

$$change\_password \triangleq change\_password_{USER} \wedge \exists MAIL \wedge \exists THE\_ANONS \wedge \exists JOURNALLED\_FILING\_SYSTEM$$

$$log\_in \triangleq log\_in_{journalled} \wedge \exists MAIL \wedge \exists JOURNALLED\_FILING\_SYSTEM \wedge \exists THE\_ANONS$$

20. Summary

This document has formally specified the requirements for an example secure system. The first part described all the components that were required in the final system and defined the simple operations upon these. The next part defined how these components were to be combined and showed how the operations on the various compoents were related to the others. This structuring of the specification makes it much easier to read and highlights the dependencies and constraints it will be necessary to enforce in the implementation. The actual set of operations that have been defined is fairly limited. However, enough operations have been specified to provide a useable system and to illustrate all the important areas.

21. References

[1] A Secure Capability Computer
S R Wiseman
Procs. IEEE Symp. Security and Privacy, Oakland CA, April 1986

[2] A Capability Approach to Multi-level Security
S R Wiseman
Procs. IFIP/Sec, Monaco, December 1986

[3] Protection and Security Mechanisms in the SMITE Capability Computer
S R Wiseman
RSRE Memorandum 4117

[4] The Z Notation: A Reference Manual
J M Spivey Draft JMS-87-12a
PRG, Oxford University

[5] The Trusted Path between SMITE and the User
S R Wiseman, P F Terry, A W Wood, C L Harrold
To appear: IEEE Symp. Security and Privacy, Oakland CA, April 1988

Thanks to Simon Wiseman, Peter Bottomley, Ruaridh Macdonald and Alf Smith for their comments and suggestions.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Report 88002		Unclassified

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
778400	Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title
FORMAL SPECIFICATION OF A SECURE DOCUMENT CONTROL SYSTEM FOR SMITE

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
Harrold C L			1988.02	72

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement
Unlimited

Descriptors (or keywords)

continue on separate piece of paper

Abstract

This Report formally describes the requirements for a demonstration of a secure electronic registry control system (Sercus) to be implemented using the security attributes of the SMITE secure capability computer.